

Design of Application Specific Processor Architectures

Rainer Leupers
RWTH Aachen University
Software for Systems on Silicon
leupers@iss.rwth-aachen.de



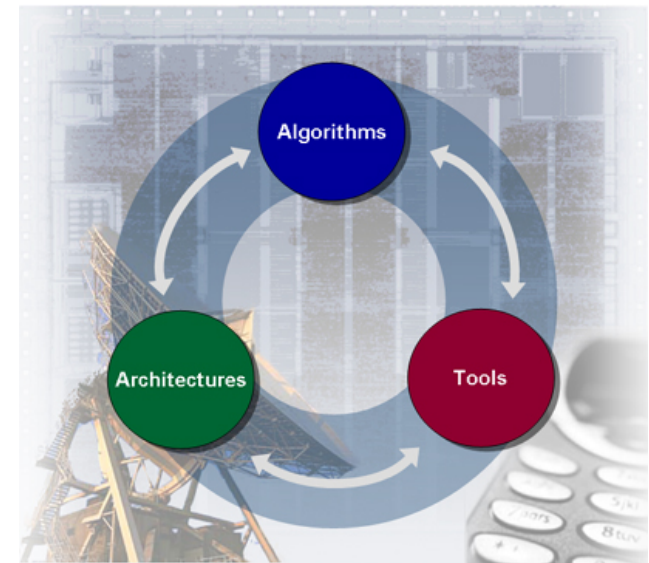
Aachen



Berlin



- RWTH Aachen is a top-ranked technical university in Germany
- ISS institute
 - 3 professors (Meyr, Ascheid, Leupers)
 - 18 Ph.D. students
 - 5 staff
- Research on wireless communication systems
 - tight industry cooperations
 - origin of several EDA spin-off companies (e.g. Cadis, Axys, LISATek)
 - SSS group focuses on embedded processor design tools



1. Introduction
2. ASIP design methodologies
3. Software tools
4. ASIP architecture design
5. Case study
6. Advanced research topics

1. Introduction

➤ Embedded systems

- Special-purpose electronic devices
- Very different from desktop computers

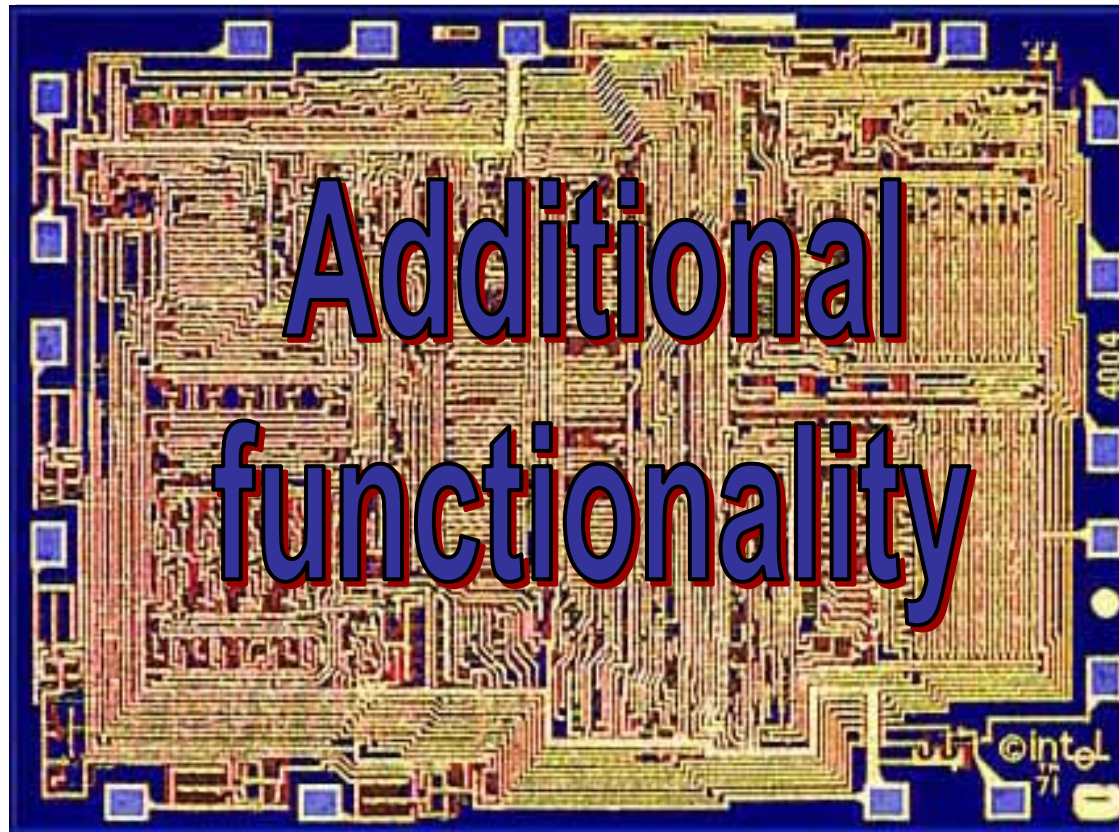
➤ Strength of European IT market

- Telecom, consumer, automotive, medical, ...
- Siemens, Nokia, Bosch, Infineon, ...

➤ New design requirements

- Low NRE cost, high efficiency requirements
- Real-time operation, dependability
- Keep pace with Moore's Law





➤ Multistandard radio

- UMTS
- GSM/GPRS/EDGE
- WLAN
- Bluetooth
- UWB
- ...



➤ Multimedia standards

- MPEG-4
- MP3
- AAC
- GPS
- DVB-H
- ...

Key issues:

- Time to market (≤ 12 months)
- Flexibility (ongoing standard updates)
- Efficiency (battery operation)

„As the performance of conventional microprocessors improves, they first meet and then exceed the requirements of most computing applications. Initially, performance is key. But eventually, other factors, like customization, become more important to the customer...“

[M.J. Bass, C.M. Christensen: The Future of the Microprocessor Business, IEEE Spectrum 2002]

design budget = (semiconductor revenue) × (% for R&D)

growth ≈ 15%

≈ 10%

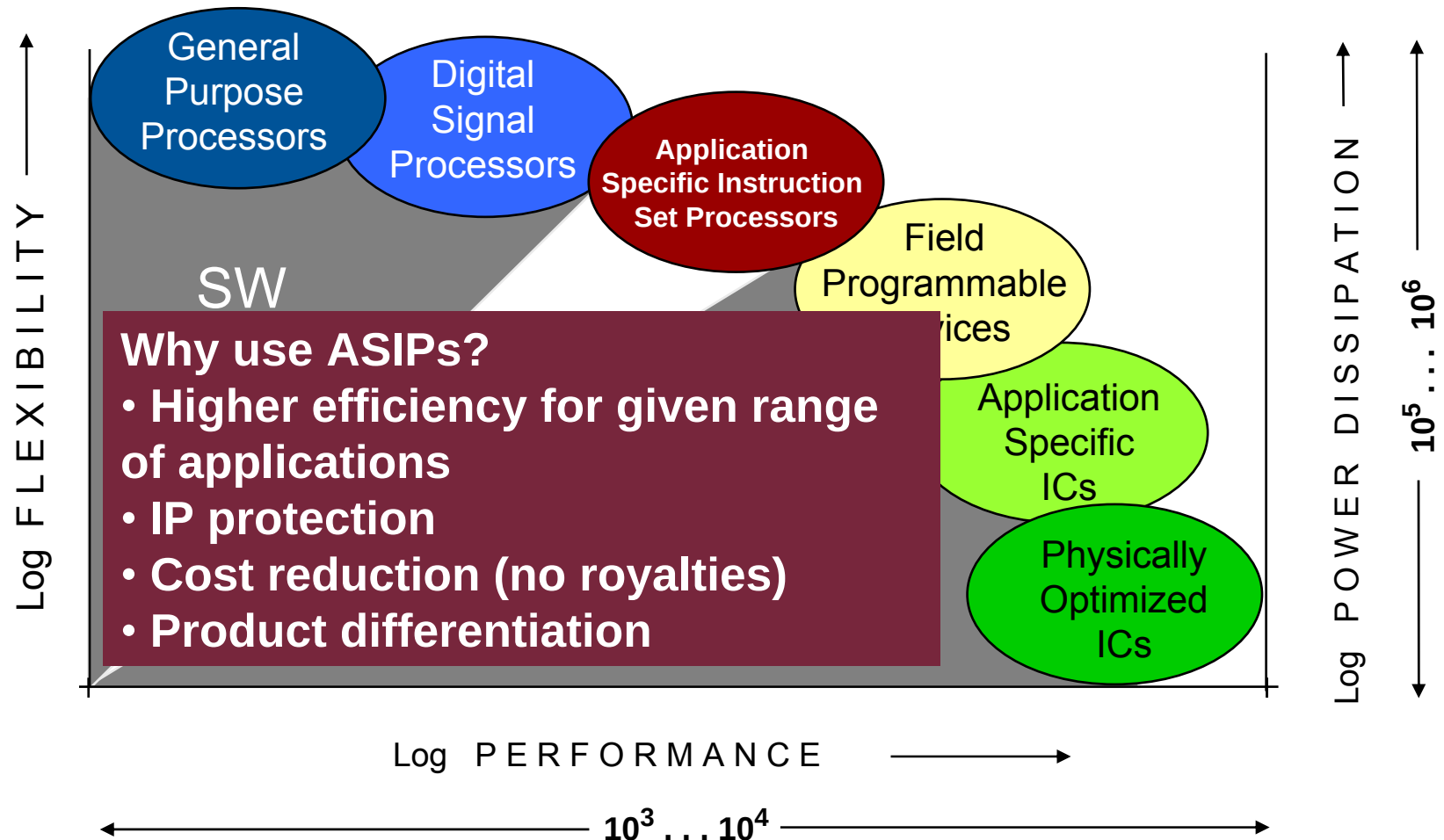
IC designs = (design budget) / (design cost per IC)

growth ≈ 15%

growth ≈ 50-100%

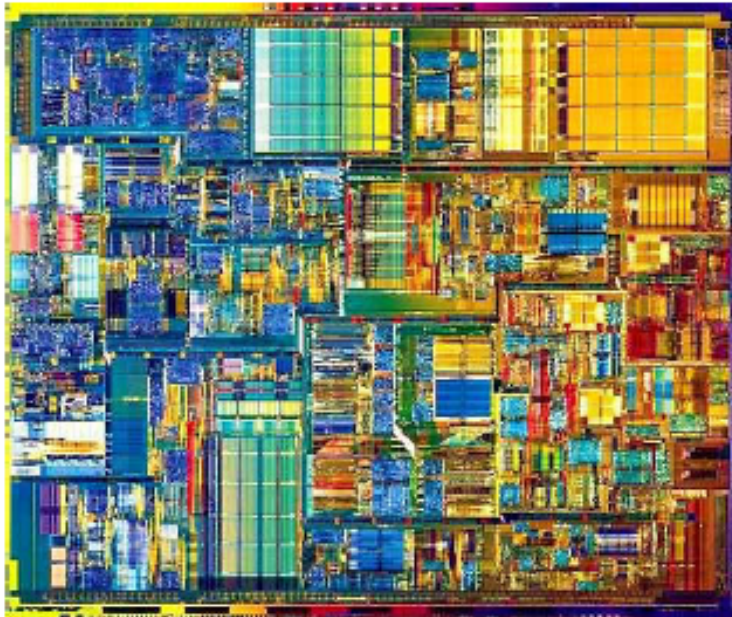
[Keutzer05]

→ Customizable application specific processors as reusable, programmable platforms



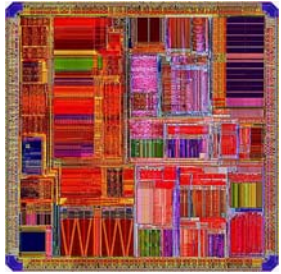
Source: T.Noll, RWTH Aachen

Intel
Pentium 4
(145mm², 50W in 0.13μ)

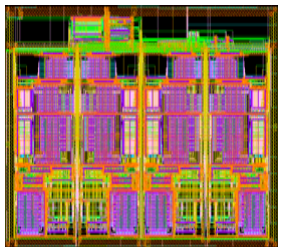


2. ASIP design methodologies

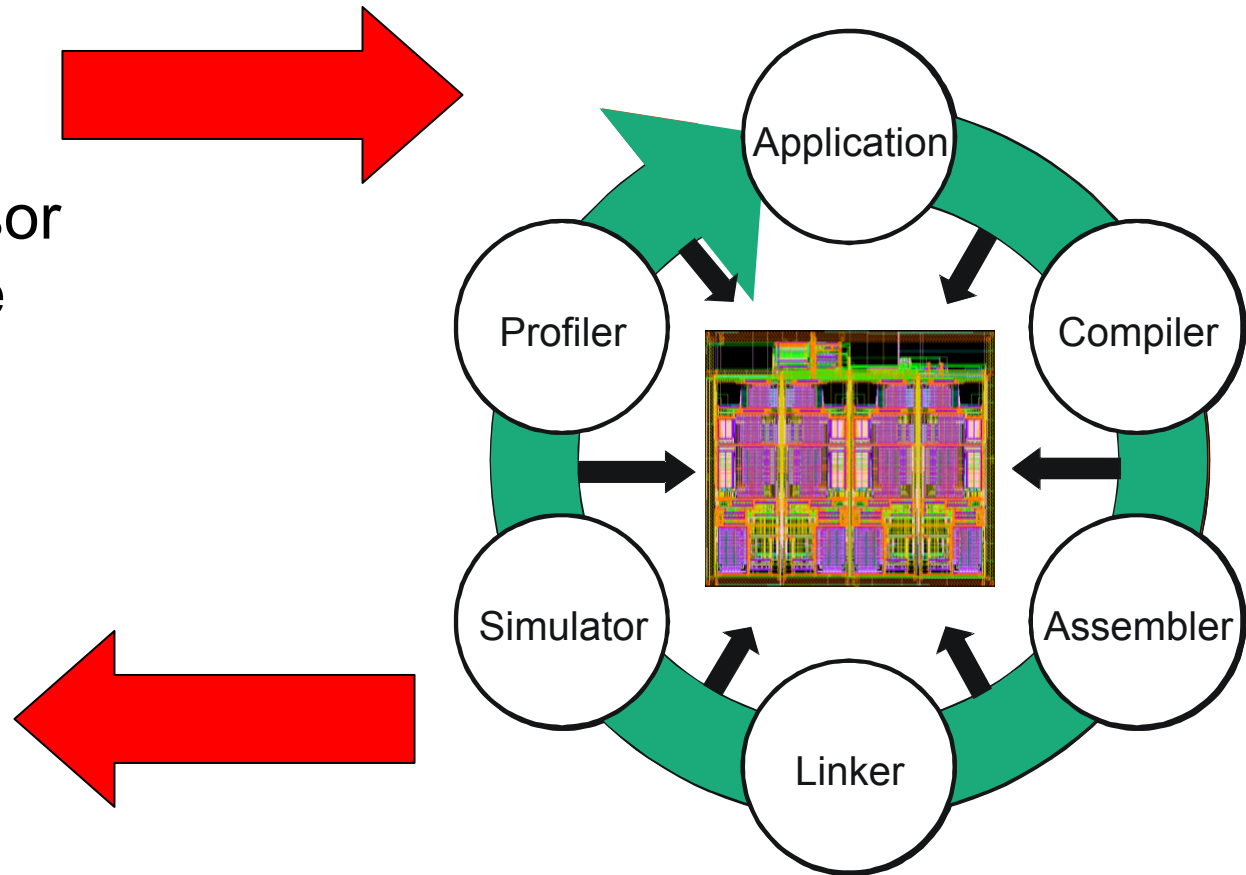
ASIP architecture exploration

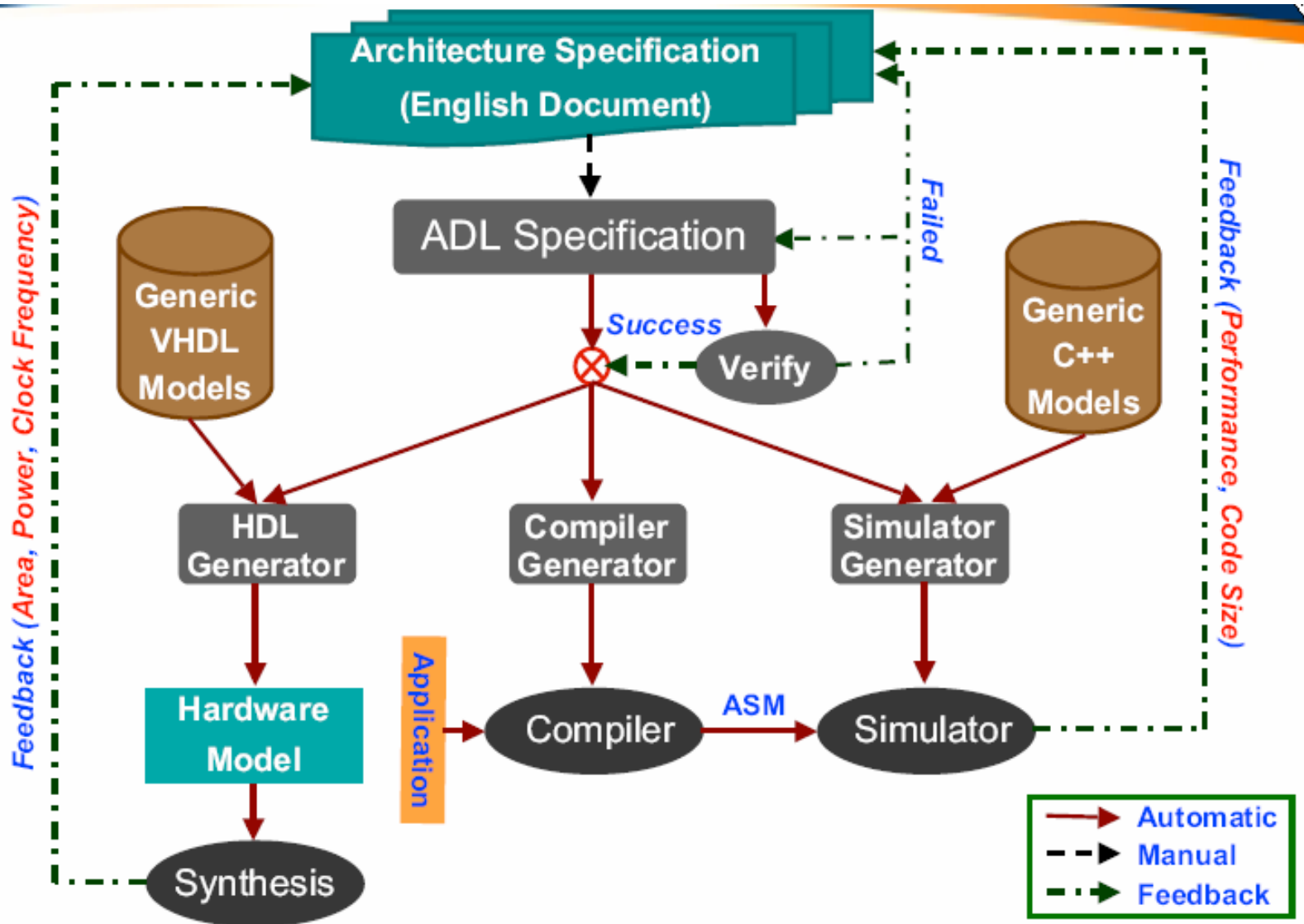


initial processor
architecture

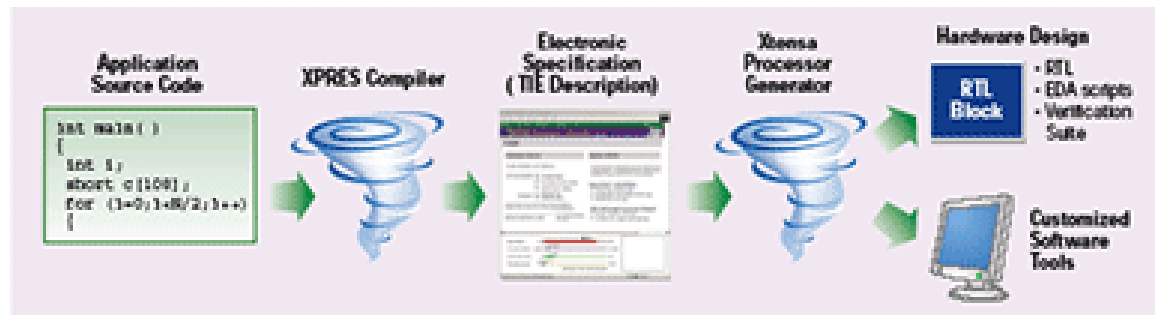


optimized
processor
architecture





From an ANSI C / C++ application the XPRES Compiler generates an optimized set of processor extensions ...



... that is reusable over a range of similar application software code.

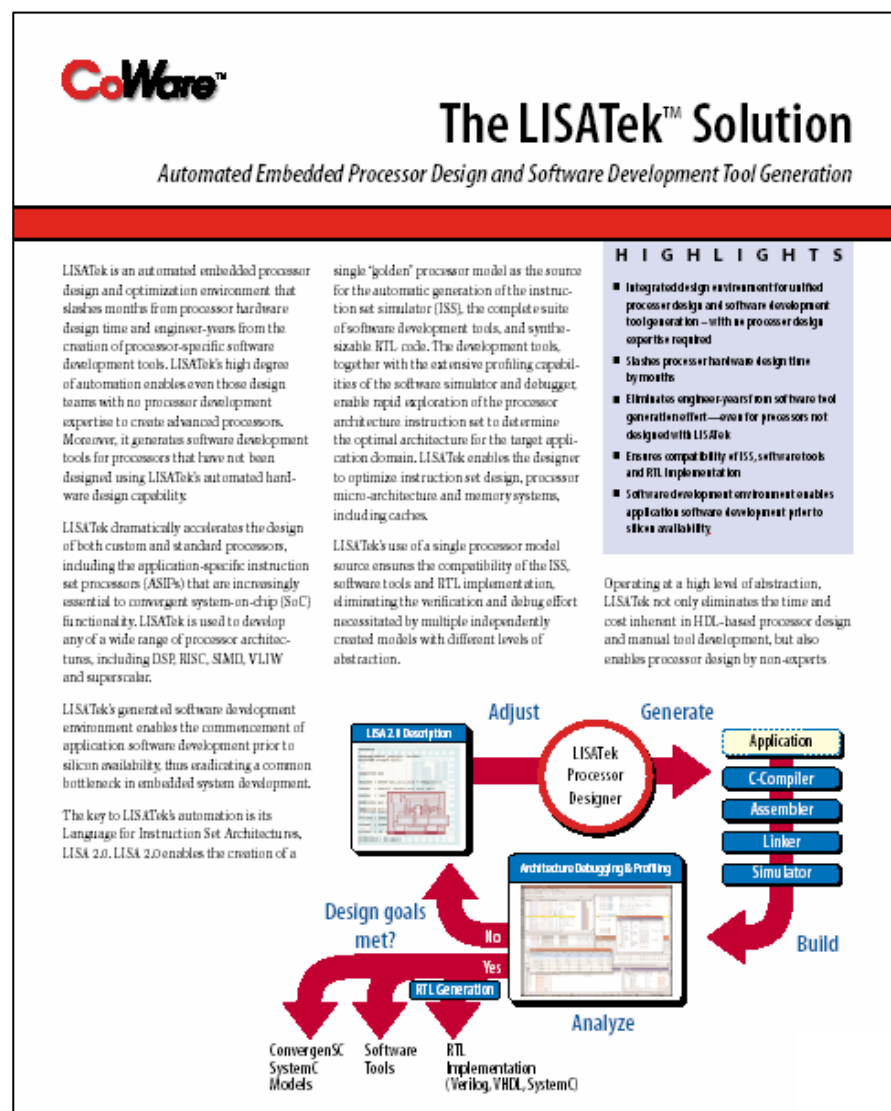


Source: Tensilica Inc.

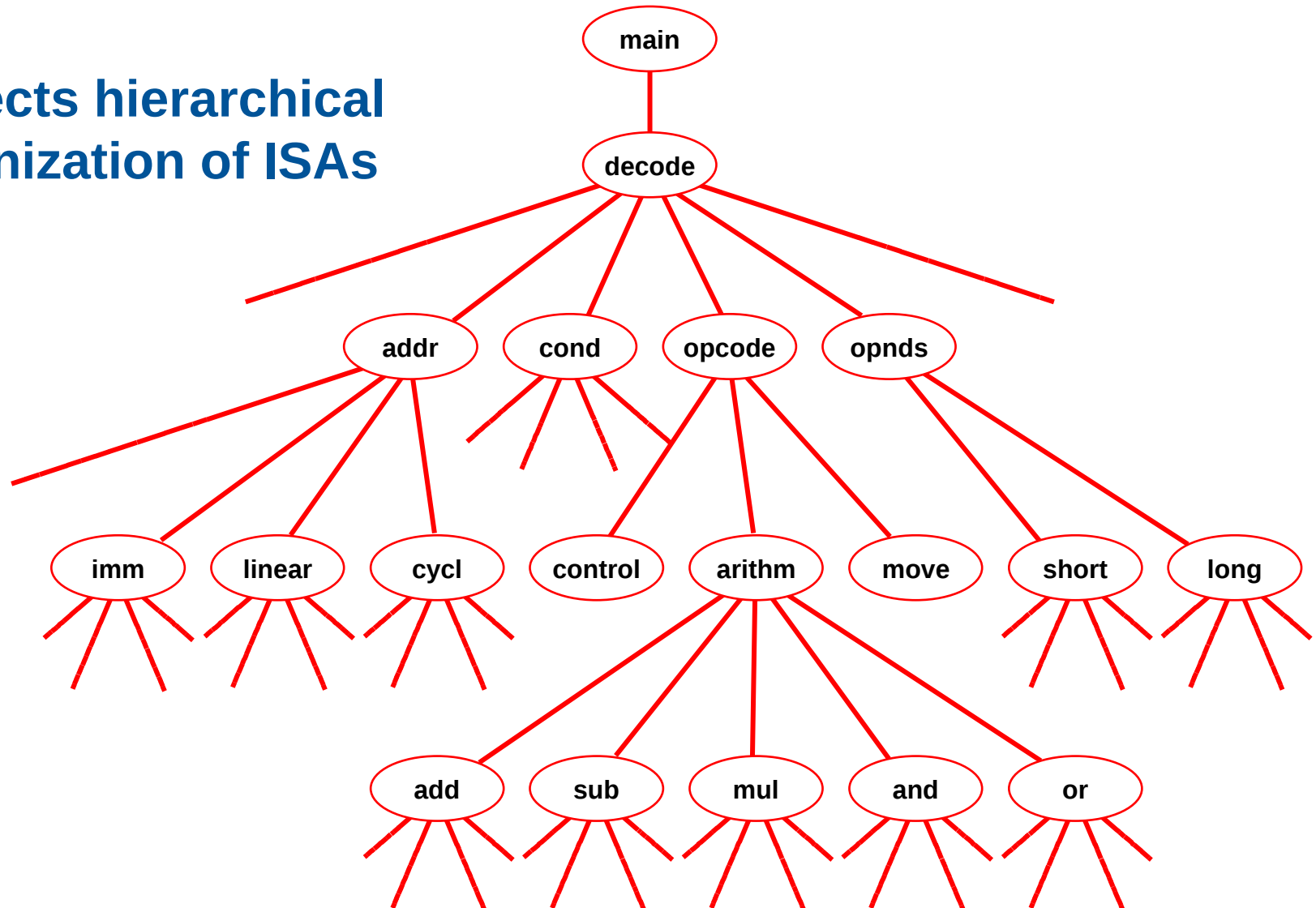
Hot spot

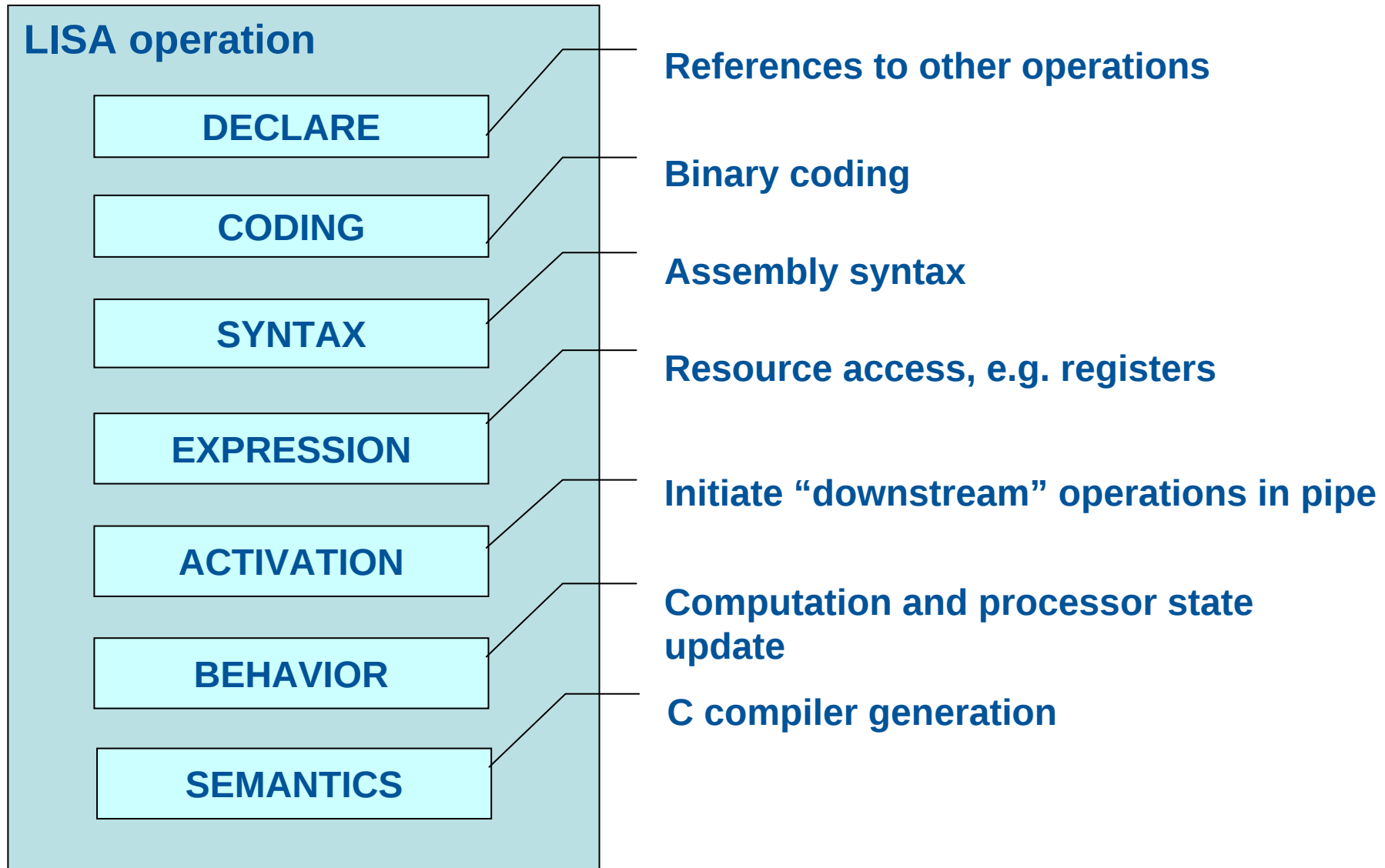


- Integrated embedded processor development environment
- Unified processor model in LISA 2.0 architecture description language (ADL)
- Automatic generation of:
 - SW tools
 - HW models



Reflects hierarchical organization of ISAs





LISA operation example

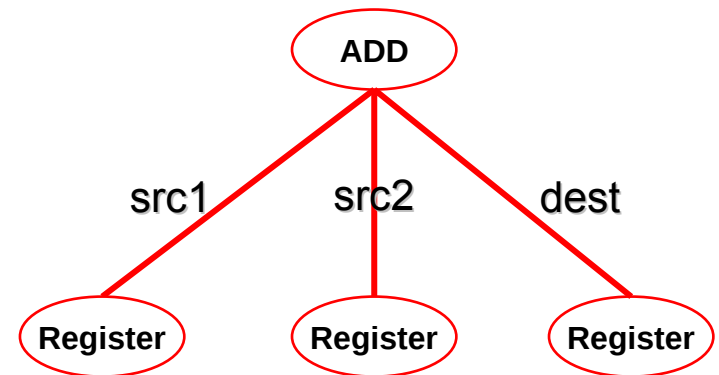
OPERATION ADD

```
{  
  DECLARE  
  {  
    GROUP src1, src2, dest = { Register }  
  }  
  CODING { 0b1011 src1 src2 dest }  
  SYNTAX { "ADD" dest "," src1 "," src2 }  
  BEHAVIOR { dest = src1 + src2; }  
}
```

← C/C++ Code

OPERATION Register

```
{  
  DECLARE  
  {  
    LABEL index;  
  }  
  CODING { index }  
  SYNTAX { "R" index }  
  EXPRESSION { R[index] }  
}
```



Exploration/debugger GUI

The screenshot displays the LISATEK RIM(TM) Processor Debugger interface. The main window shows the 'LISA Operation Profile' with a table of resource utilization. A 'LISA Pipeline Profile' window is also visible, showing pipeline stages (PF, FE, DC, AC, RD, EX) and their respective counts. A 'Disassembly' window shows the assembly code for the current instruction. A 'Resource Value' window shows the current values of various resources. A 'Name In Out' window shows the input and output values for various resources. A 'Code coverage analysis' window shows the coverage of various code blocks.

Resource	Value
pc	00001565
REA	00000000
RSA	00000000
BRC	00000000
BK	00000000
RPTC	00000000
SP	00000efb
AR7	00000000
AR6	00000000
AR5	00000000
AR4	00000000
AR3	00000000
AR2	00000af0
AR1	00000889
AR0	00000000
TRN	00000000
T	00000000
IFR	00000000
IMR	00000000
PMST	0000ffc0
ST1_reg	00006900
ST0	00001800
BL	00000001
BH	00000000
BG	00000000
AL	0000005f
AH	00000000
AG	00000000

Name	Calls	Calls/Total	Calls/Max	Calls/Max	Stz
Access	146697	7.34%	55.06%		0
SmemShortRe	44029	2.20%	16.53%		0
PrgCntlInsn	26378	1.32%	9.90%		47
SmemShortWr	25375	1.27%	9.52%		0

Address	Instruc	Disassembly	Loop	Control	Relative	Re
[0000155a]	7102	MVDR _outp:0x0002, 0x001	1858	0.70		
[0000155c]	4811	LDM 0x0011, A	929	0.35		
[0000155d]	6000	ADD #_lflags0:0x0001, 0	1858	0.70		
[0000155f]	8002	STL A, _outp:0x0002	929	0.35		
[00001560]	1006	LD _valpred0:0x0006, A	929	0.35		
[00001561]	8081	STL A, *AR1	929	0.35		
[00001562]	6b0e	ADDM #0xffff, 0x000e	1858	0.70		
[00001563]	1006	LD _valpred0:0x0006, A	929	0.35		
[00001564]	1006	LD _valpred0:0x0006, A	4640	1.74		
[00001565]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001566]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001567]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001568]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001569]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000156a]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000156b]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000156c]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000156d]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000156e]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000156f]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001570]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001571]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001572]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001573]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001574]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001575]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001576]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001577]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001578]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001579]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000157a]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000157b]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000157c]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000157d]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000157e]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000157f]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001580]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001581]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001582]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001583]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001584]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001585]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001586]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001587]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001588]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001589]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000158a]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000158b]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000158c]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000158d]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000158e]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000158f]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001590]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001591]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001592]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001593]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001594]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001595]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001596]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001597]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001598]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001599]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000159a]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000159b]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000159c]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000159d]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000159e]	1006	LD _valpred0:0x0006, A	0	0.00		
[0000159f]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015a0]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015a1]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015a2]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015a3]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015a4]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015a5]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015a6]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015a7]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015a8]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015a9]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015aa]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ab]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ac]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ad]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ae]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015af]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015b0]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015b1]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015b2]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015b3]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015b4]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015b5]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015b6]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015b7]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015b8]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015b9]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ba]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015bb]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015bc]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015bd]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015be]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015bf]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015c0]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015c1]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015c2]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015c3]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015c4]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015c5]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015c6]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015c7]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015c8]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015c9]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ca]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015cb]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015cc]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015cd]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ce]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015cf]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015d0]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015d1]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015d2]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015d3]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015d4]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015d5]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015d6]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015d7]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015d8]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015d9]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015da]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015db]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015dc]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015dd]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015de]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015df]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015e0]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015e1]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015e2]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015e3]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015e4]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015e5]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015e6]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015e7]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015e8]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015e9]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ea]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015eb]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ec]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ed]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ee]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ef]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015f0]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015f1]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015f2]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015f3]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015f4]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015f5]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015f6]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015f7]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015f8]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015f9]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015fa]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015fb]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015fc]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015fd]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015fe]	1006	LD _valpred0:0x0006, A	0	0.00		
[000015ff]	1006	LD _valpred0:0x0006, A	0	0.00		
[00001600]	1006					

➤ DSP:

- Texas Instruments
TMS320C54x
- Analog Devices
ADSP21xx
- Motorola 56000

➤ RISC:

- MIPS32 4K
- ESA LEON SPARC 8
- ARM7100
- ARM926

• VLIW:

- Texas Instruments
TMS320C6x
- STMicroelectronics
ST220

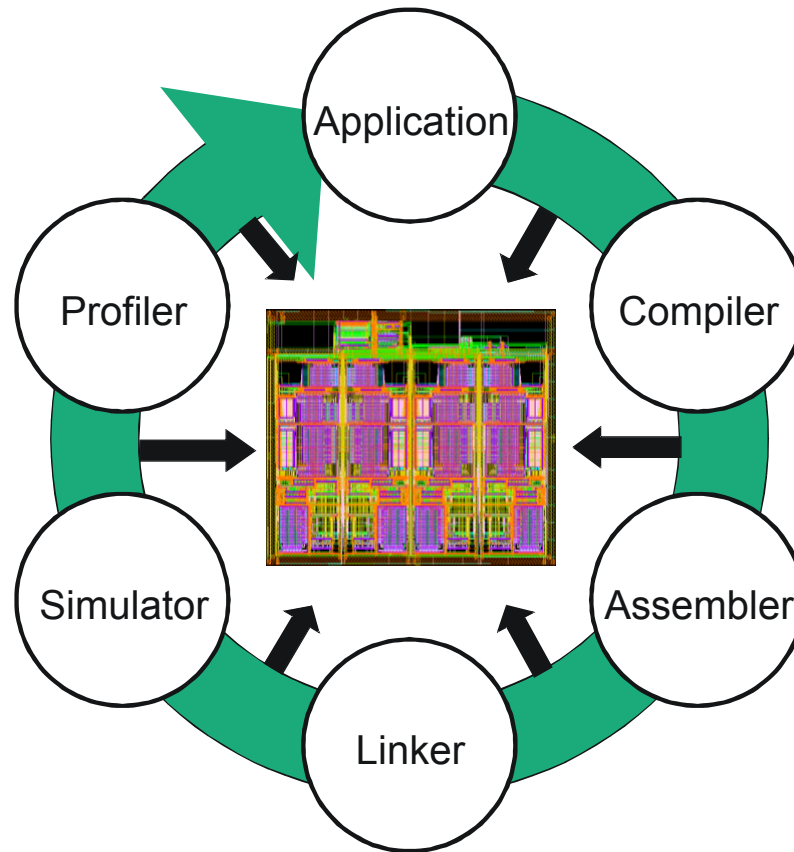
• μ C:

- MHS80C51

• ASIP:

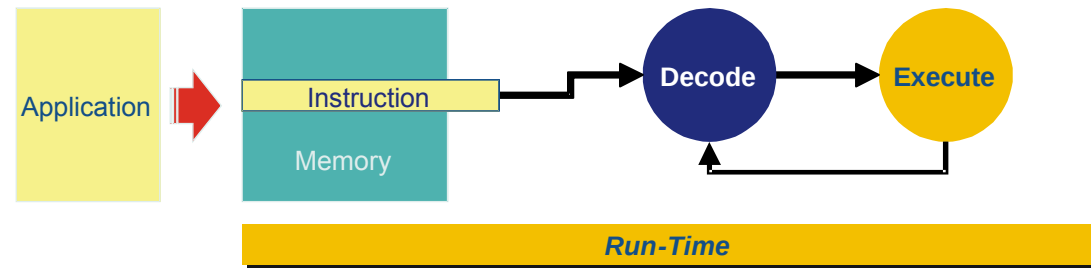
- Infineon PP32 NPU
- Infineon ICore
- MorphICs DSP

3. Software tools



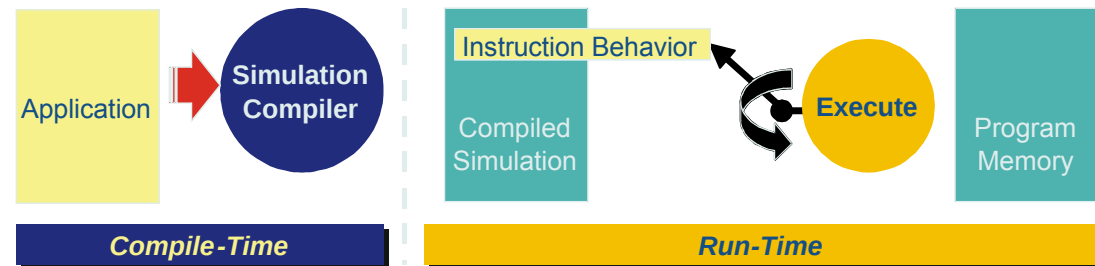
Interpretive:

- flexible
- slow (~ 100 KIPS)



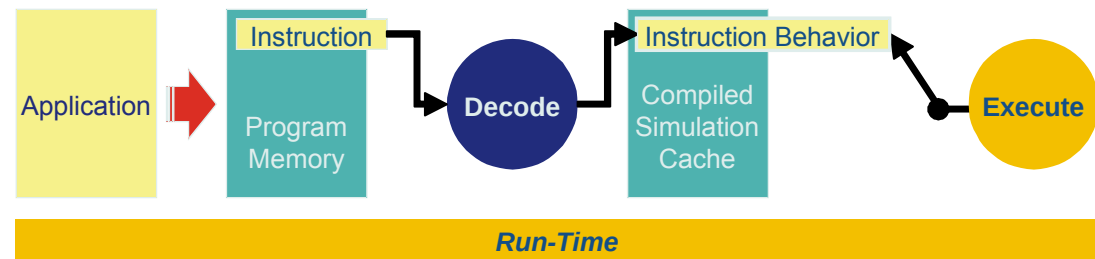
Compiled:

- fast (> 10 MIPS)
- inflexible
- high memory consumption

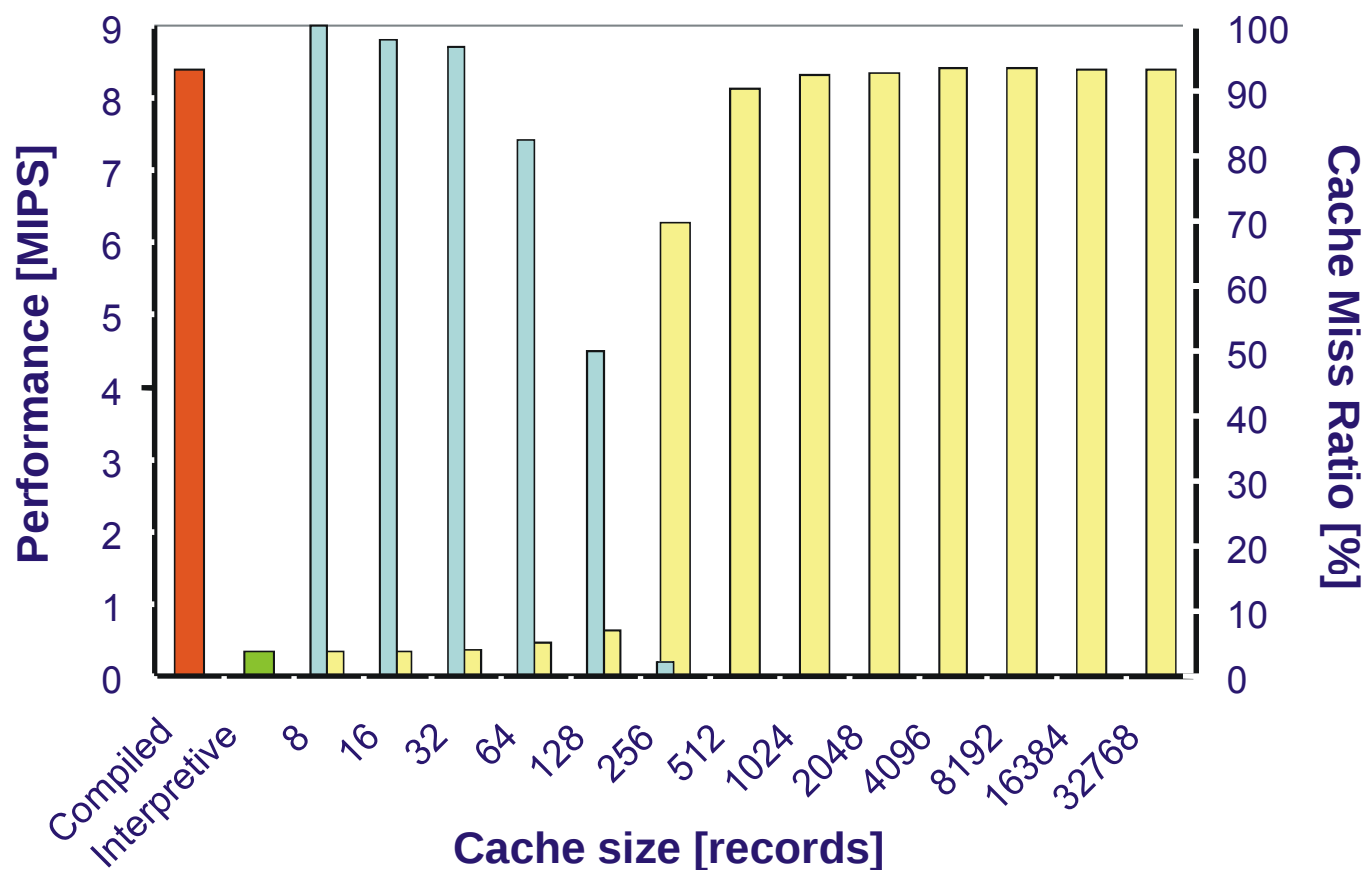


JIT-CCS™:

- „just-in-time“ compiled
- SW simulation cache
- fast and flexible

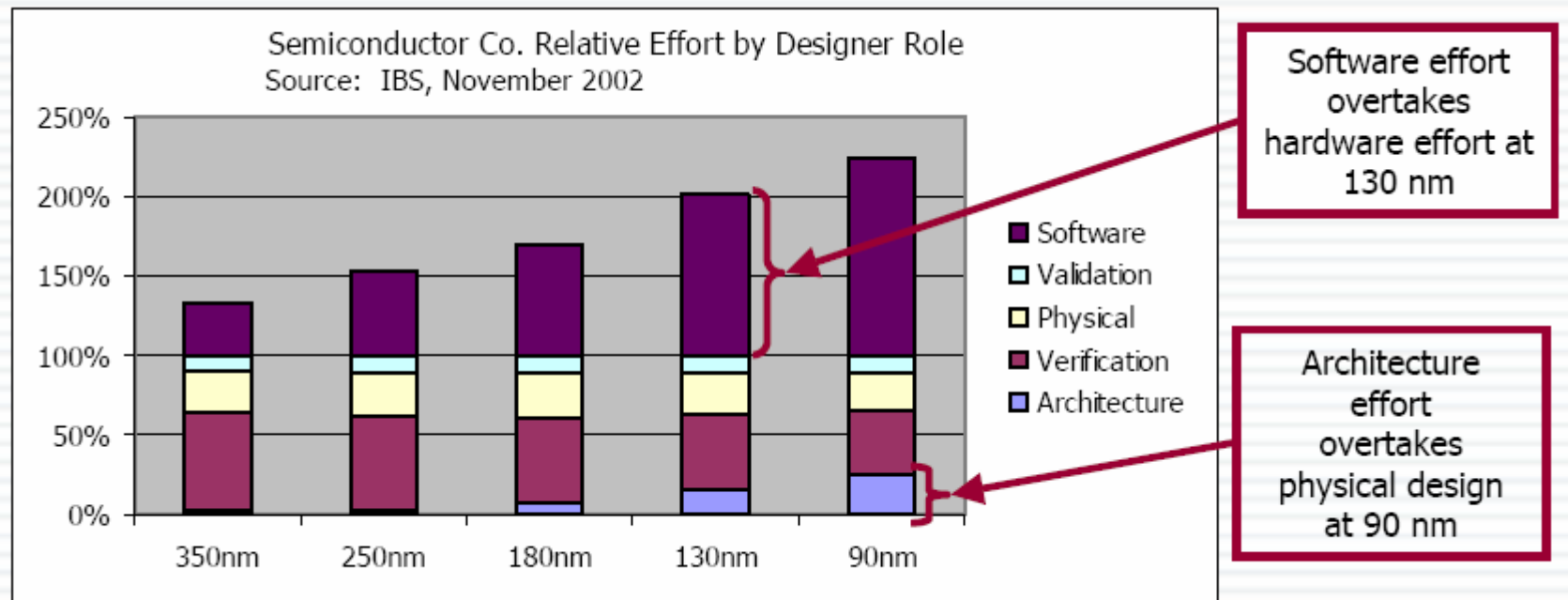


- Dependent on simulation cache size
- 95% of compiled simulation performance @ 4096 cache blocks (10% memory consumption of compiled sim.)
- Example: ST200 VLIW DSP



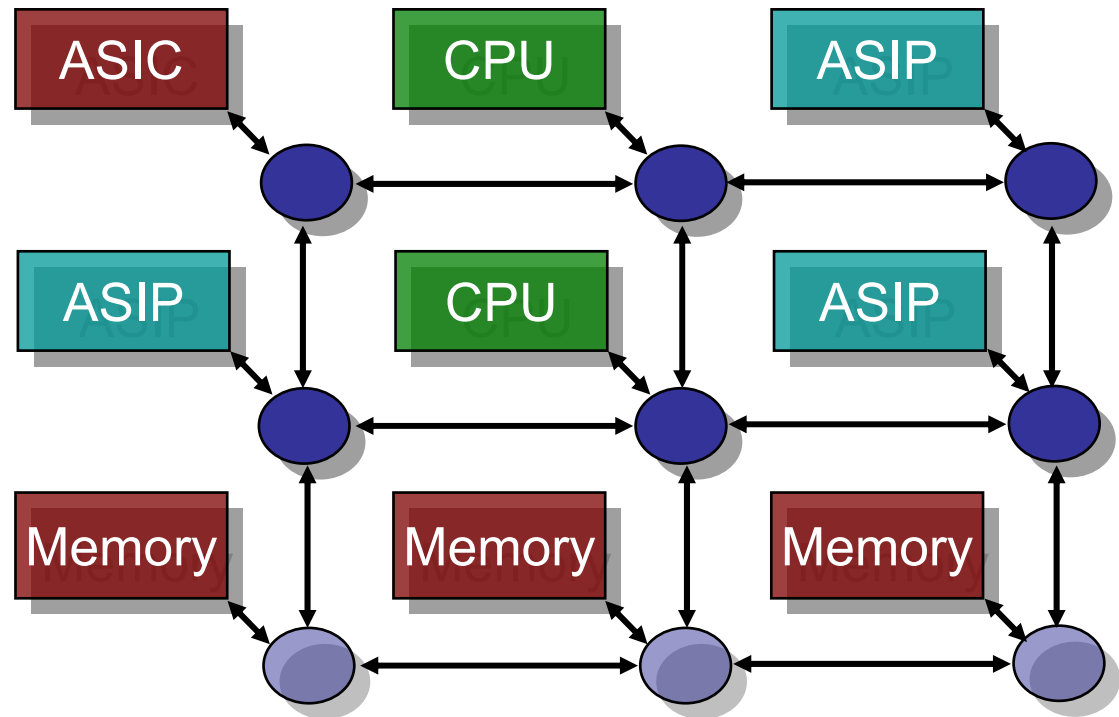
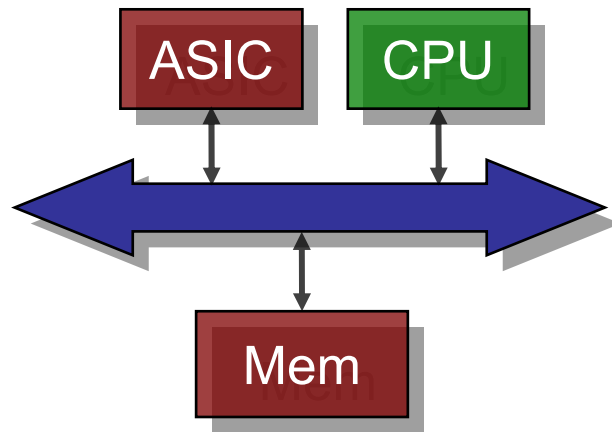
Why care about C compilers?

- Embedded SW design becoming predominant manpower factor in system design
- Cannot develop/maintain millions of code lines in assembly language
- Move to high-level programming languages



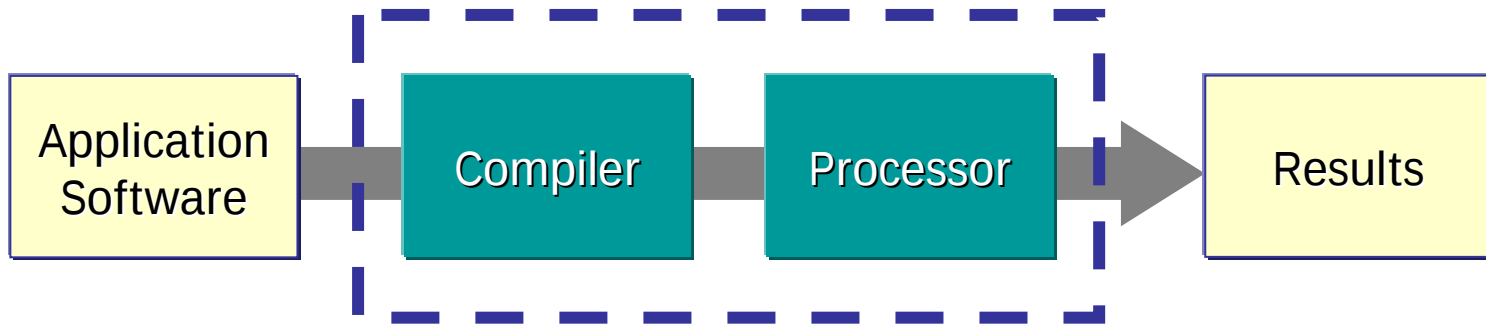
Why care about compilers?

- Trend towards heterogeneous multiprocessor systems-on-chip (MPSoC)
- Customized application specific instruction set processors (ASIPs) are key MPSoC components
- How to achieve efficient compiler support for ASIPs?



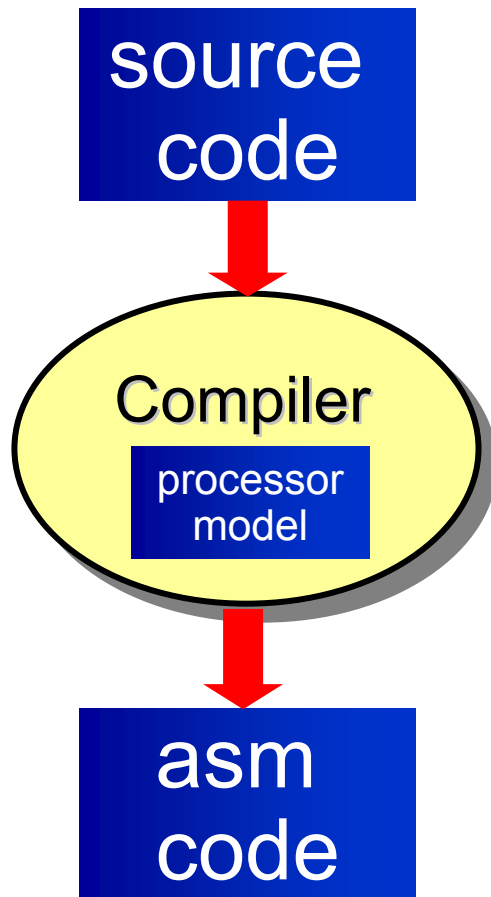
„Compiler/Architecture Co-Design“

- Efficient C-compilers **cannot** be designed for **ARBITRARY** architectures!

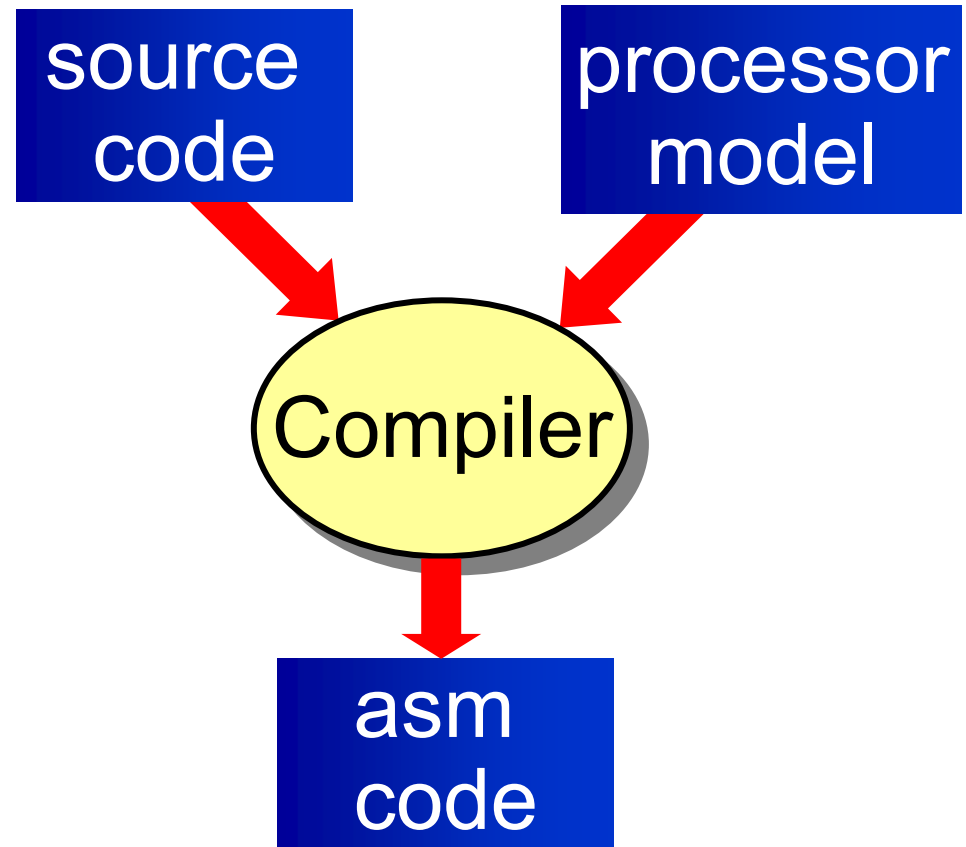


- Compiler and processor form a **UNIT** that needs to be optimized!
- “Compiler-friendliness” needs to be taken into account during the architecture exploration!

Classical compiler



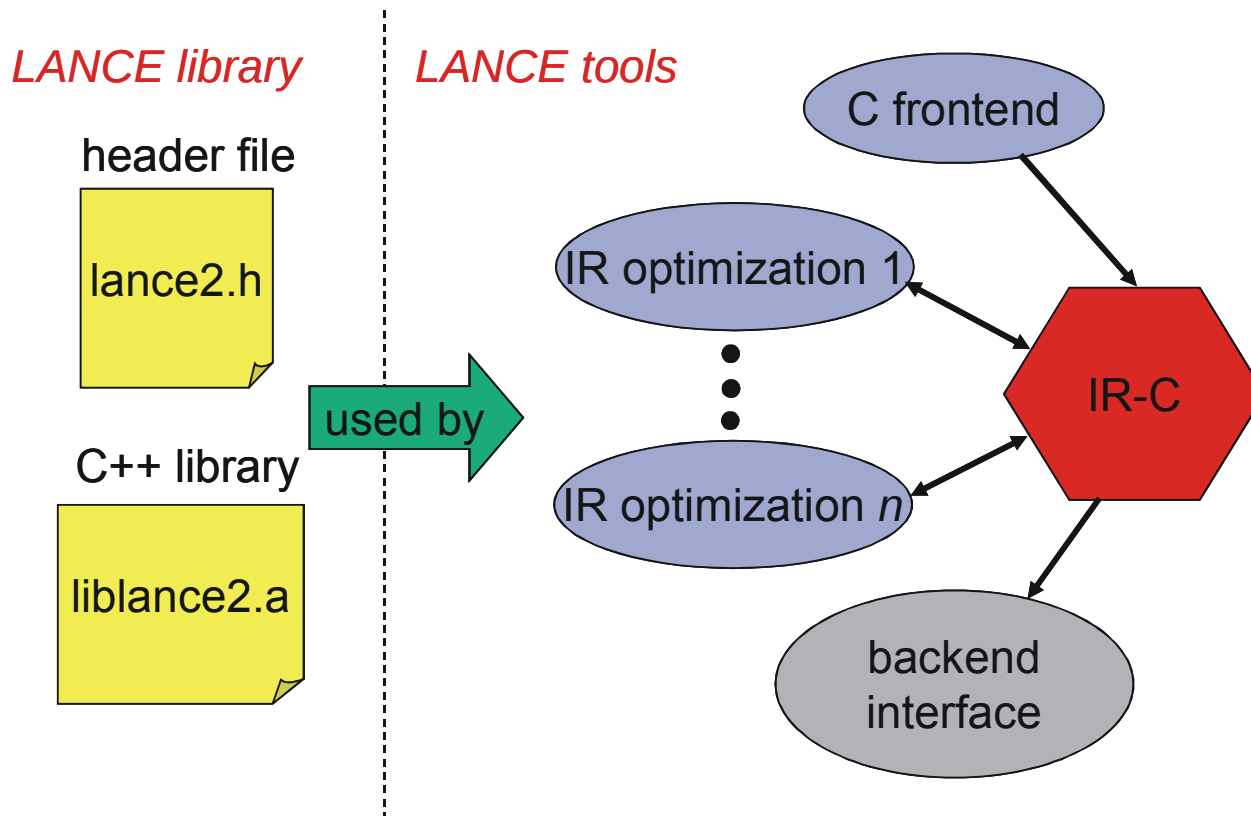
Retargetable compiler



- Probably the **most widespread** retargetable compiler
- Mostly used as a native Unix/Linux compiler, but may operate as a **cross-compiler**, too
- Support for C/C++, Java, and other languages
- Comes with comprehensive **support software**, e.g. runtime and standard libraries, debug support
- Portable to new architectures by means of **machine description file** and C support routines

"The main goal of GCC was to make a good, fast compiler for machines in the class that the GNU system aims to run on: 32-bit machines that address 8-bit bytes and have several general registers. Elegance, theoretical power and simplicity are only secondary."

- Modular retargetable C compiler system
- www.lancecompiler.com



Executable C based IR in the LANCE compiler

C source code

```
void f()  
{  
    int i,A[10];  
    i = A[2]++  
        > 1 ?  
        2 : 3;  
}
```

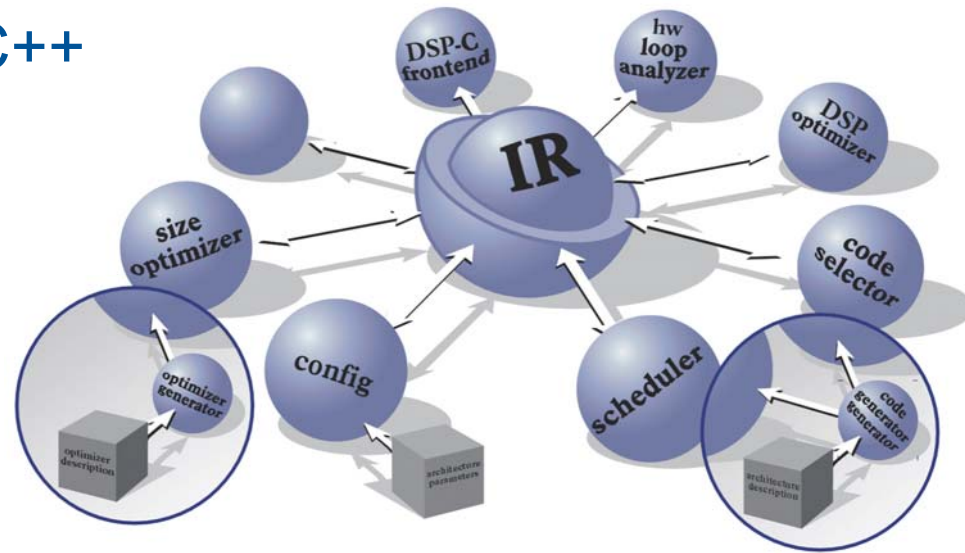
```
int A[10];  
char *t1,*t3;  
int i,t2,t5,t6,t7,t8;  
int *t4;
```

```
t3 = (char *)A;    // cast base to char*  
t2 = 2 * 4;        // compute offset  
t1 = t3 + t2;      // compute eff addr  
t4 = (int *)t1;    // cast back to int*  
t5 = *t4;          // load value from memory
```

```
t6 = t5 + 1;       // increment  
*t4 = t6;          // store back into A[2]
```

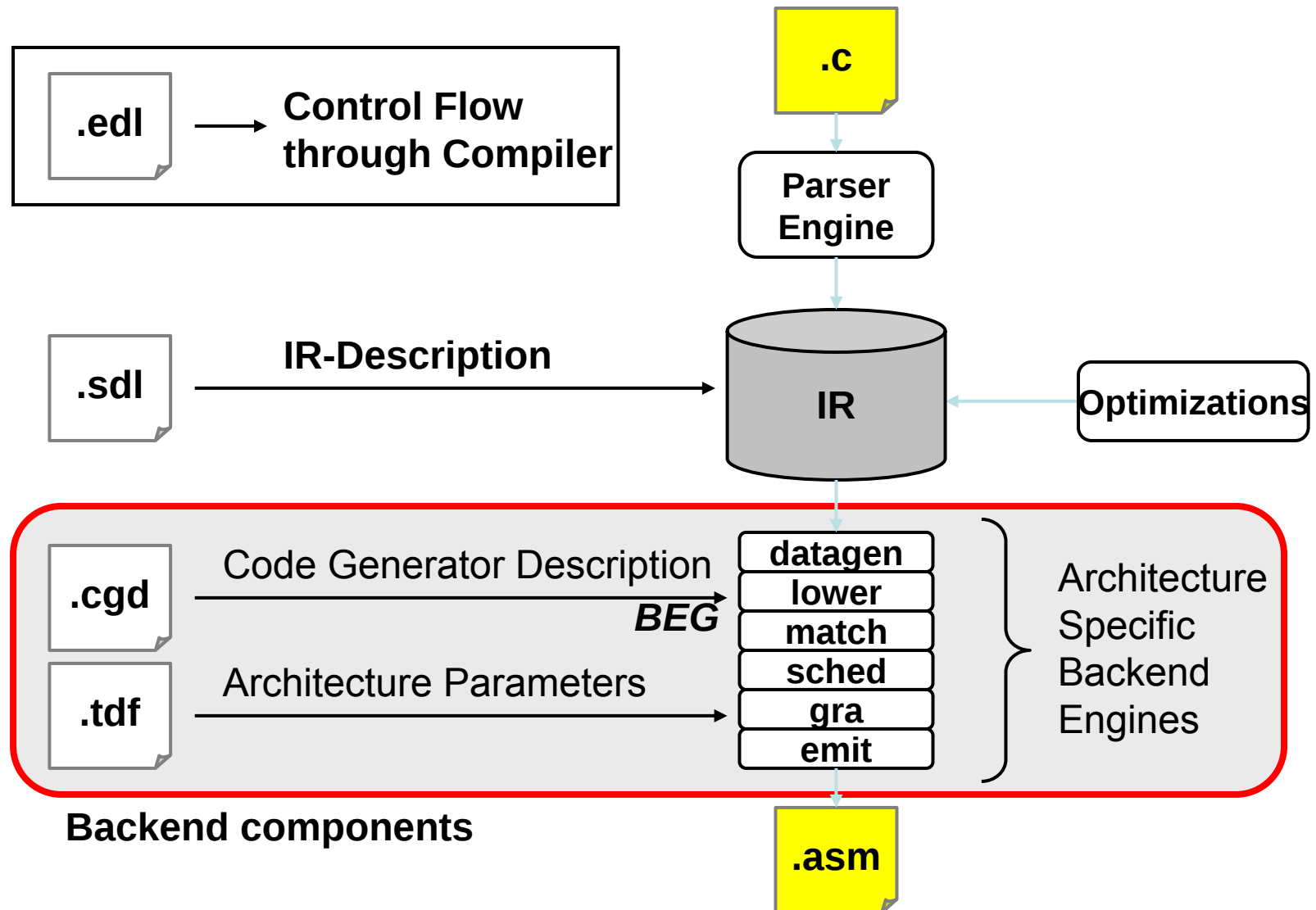
```
t7 = t5 > 1;       // compare  
if (t7) goto L1;   // jump if >  
t8 = 3;            // load 3 if <=  
goto L2;           // goto join point  
L1: t8 = 2;         // load 2 if >  
L2: i = t8;        // move result into i
```

- Universal retargetable C/C++ compiler
- Extensible intermediate representation (IR)
- Modular compiler organization
- Generator (BEG) for code selector, register allocator, scheduler



© ACE - Associated
Compiler Experts

ACE CoSy system structure



LISA processor model

```
SYNTAX {  
  "ADD" dst, src1, src2  
}  
  
CODING {  
  0b0010 dst src1 src2  
}  
  
BEHAVIOR {  
  ALU_read (src1, src2);  
  ALU_add ();  
  Update_flags ();  
  writeback (dst);  
}  
  
SEMANTICS {  
  src1 + src2 → dst;  
}  
...
```

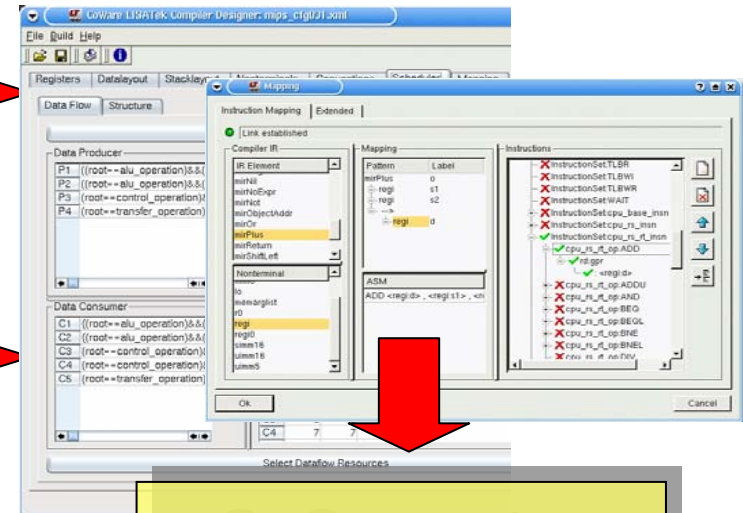


Autom. analyses

Manual refinement

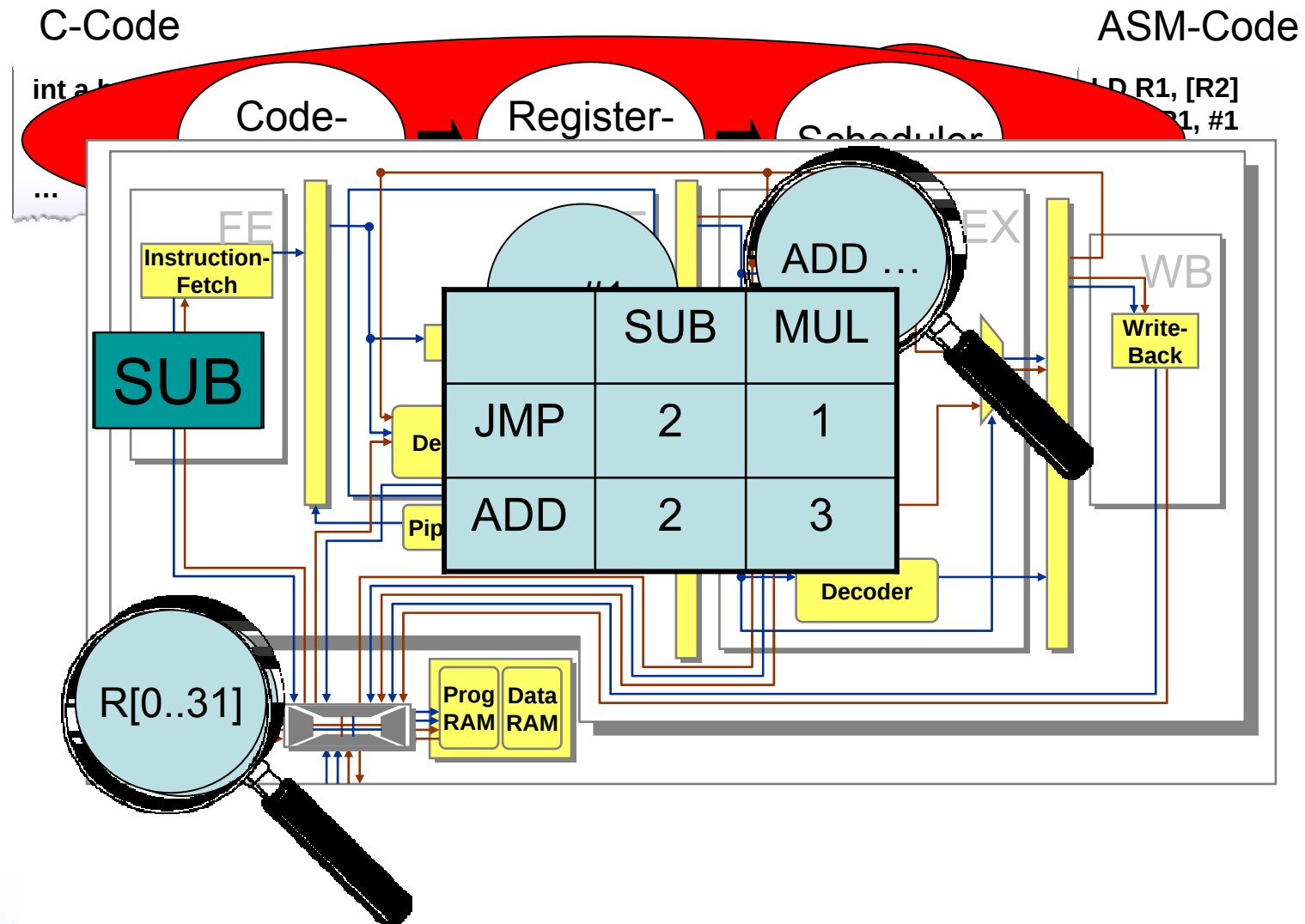


GUI



CoSy system

C Compiler

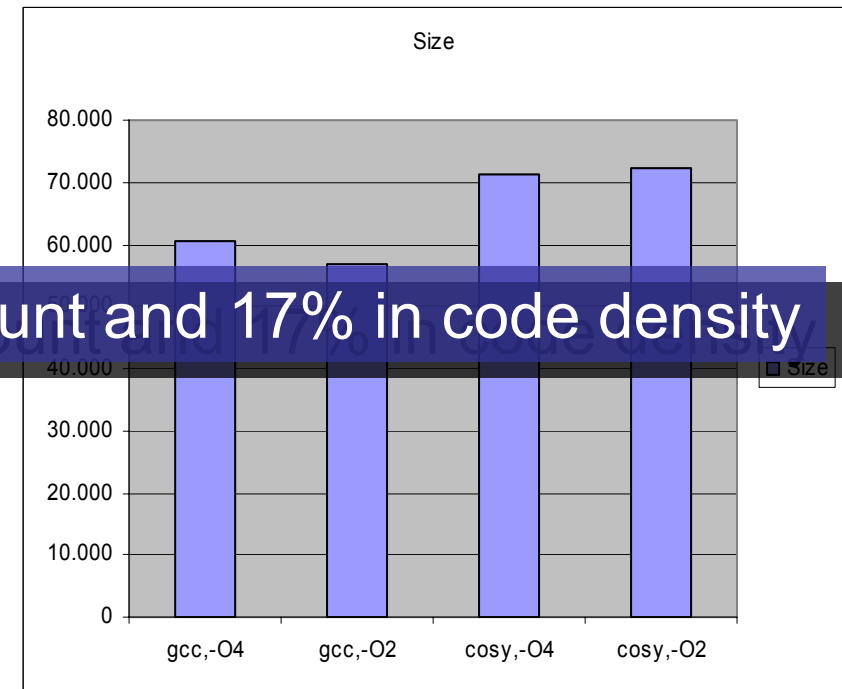
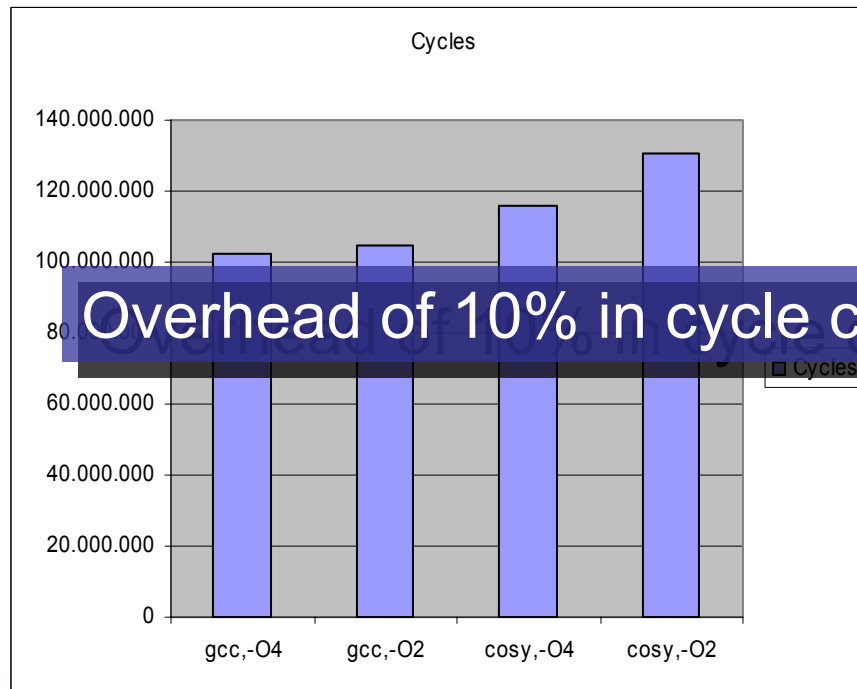


Compiled code quality: MIPS example

- LISATek generated C-Compiler
- Out-of-the-box C-Compiler
- No manual optimizations
- Development time of model approx. 2 weeks

gcc C-Compiler

- gcc with MIPS32 4kc backend
- Used by most MIPS users
- Large group of developers, several man-years of optimization



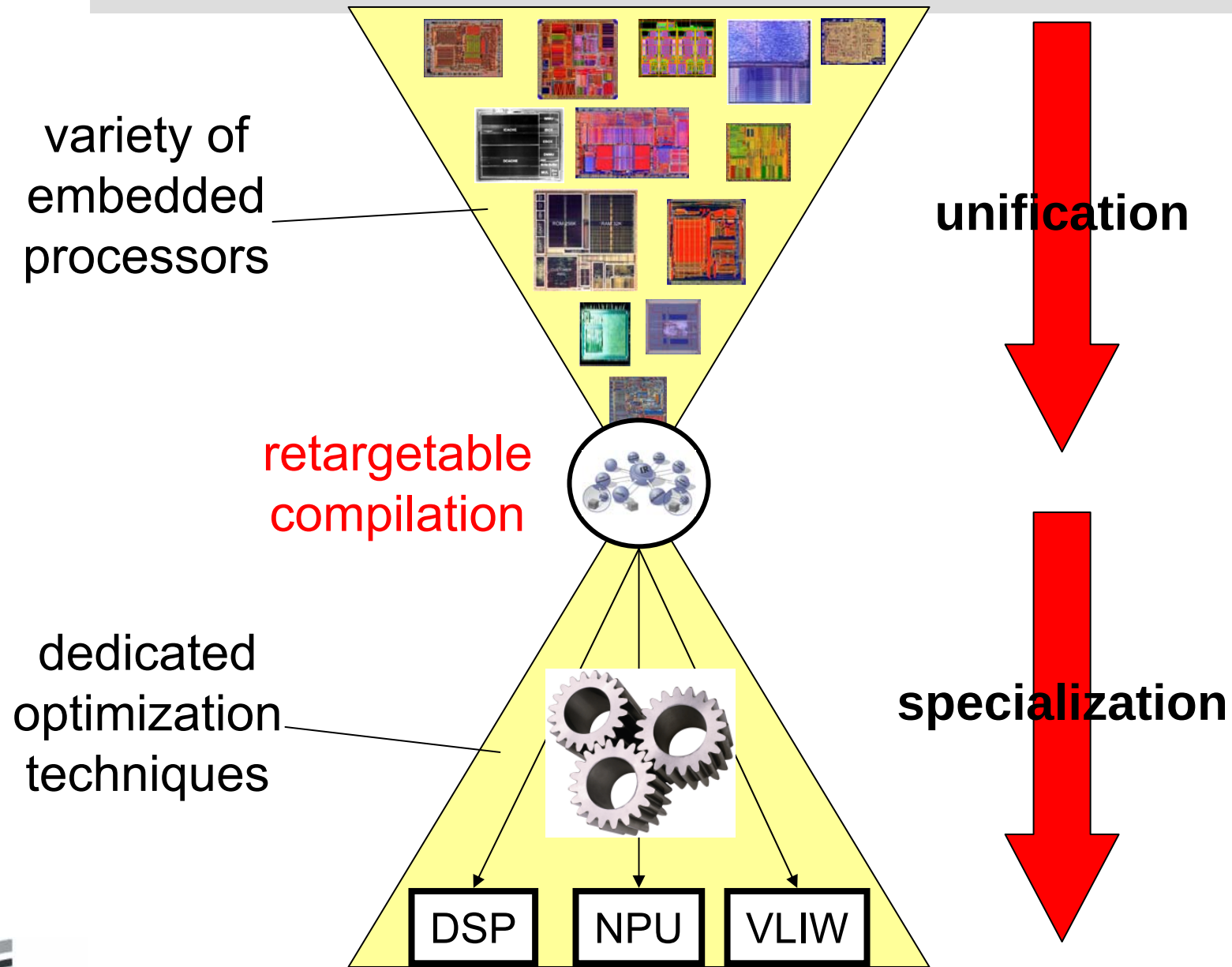
Overhead of 10% in cycle count and 17% in code density

➤ Compilers for embedded processors have to generate extremely efficient code

- Code size:
 - » system-on-chip
 - » on-chip RAM/ROM
- Performance:
 - » real-time constraints
- Power/energy consumption:
 - » heat dissipation
 - » battery lifetime

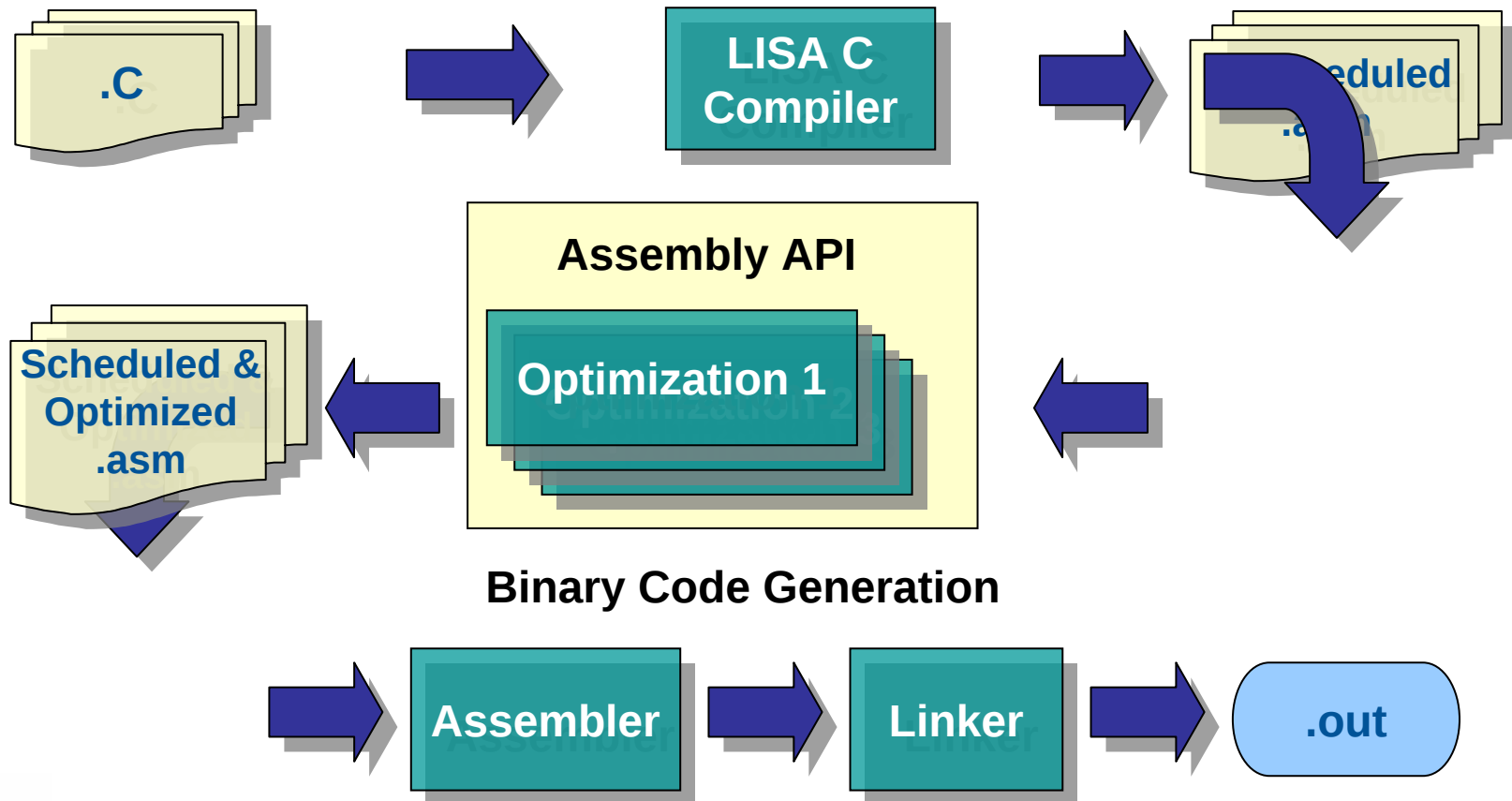
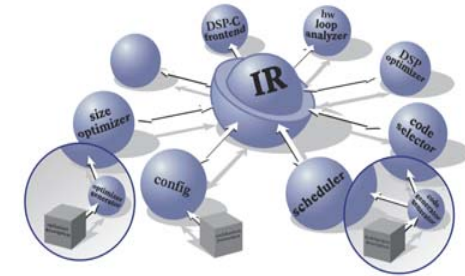


Compiler flexibility/code quality trade-off



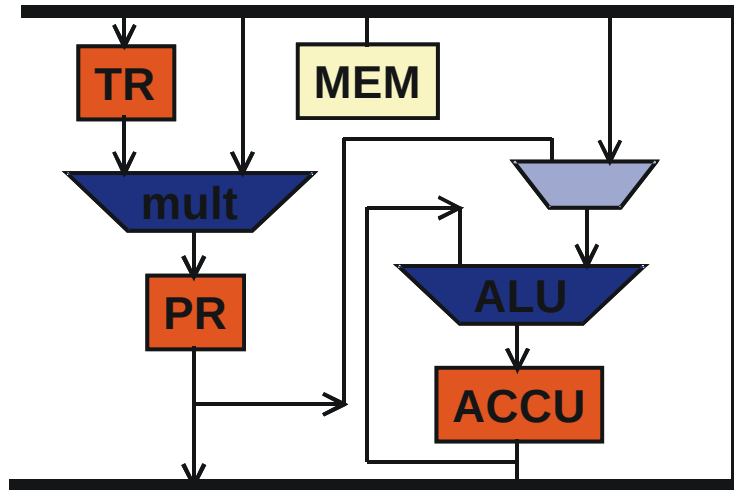
Adding processor-specific code optimizations

- High-level (compiler IR)
 - Enabled by CoSy's engine concept
- Low-level (ASM):

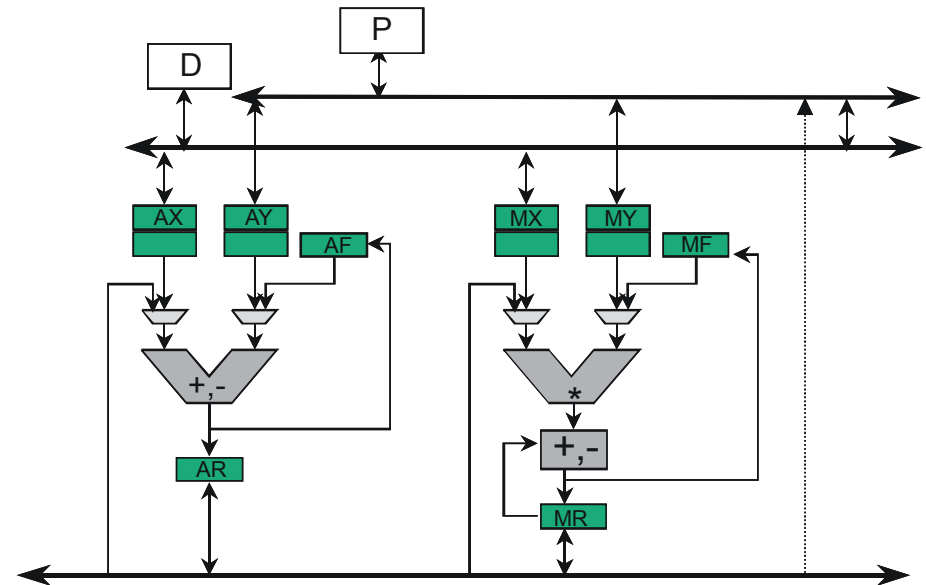


Embedded processors are not compiler-friendly

- Designed for efficiency
- E.g. fixed-point DSP data paths:



TI C25

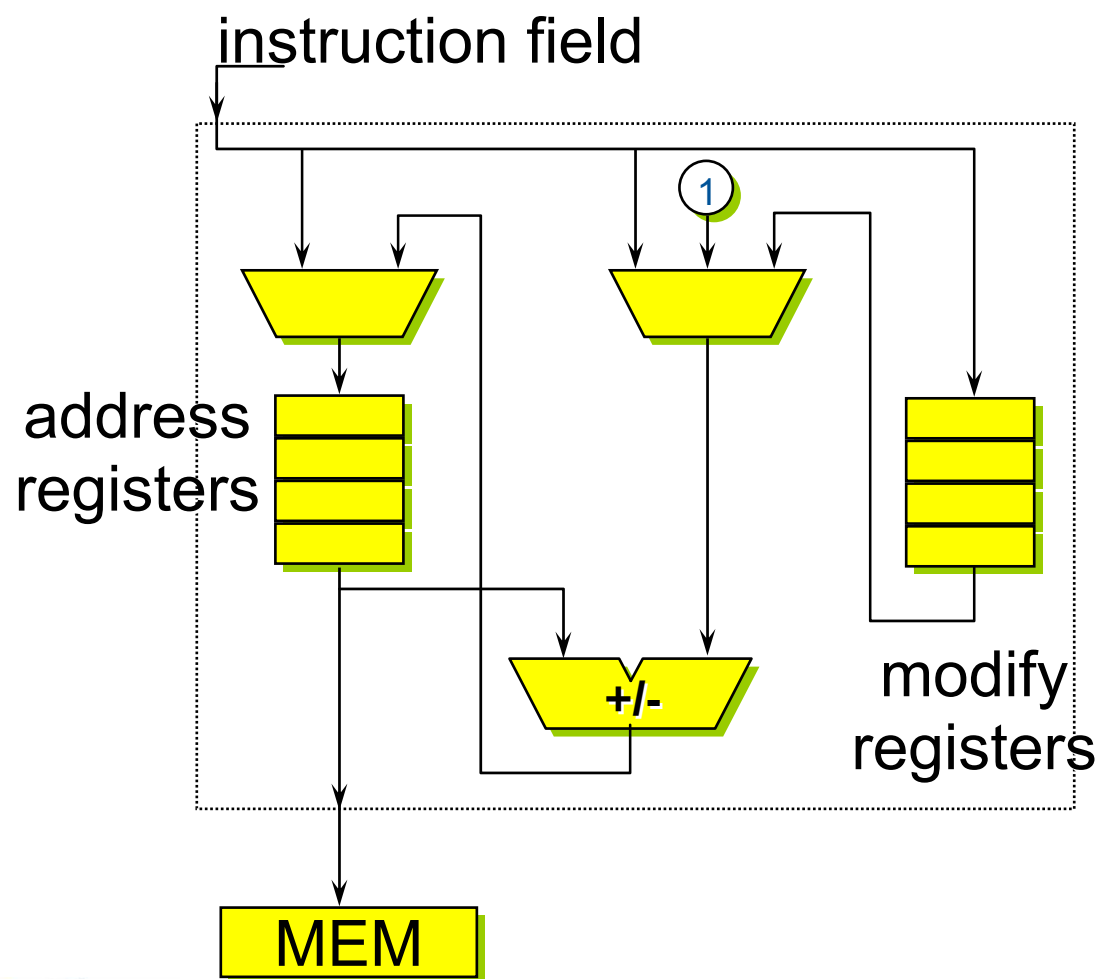


ADSP210x

- Special purpose registers, constrained parallelism, ...
- Challenge for compilers that usually prefer orthogonal „compiler-friendly“ architectures

DSP address code optimization

- Based on **address generation unit (AGU)** support
- Address arithmetic in parallel to central data path



Support for
auto-increment and
auto-modify

Examples:

- TI C2x/C5x
- Motorola 56000
- ADSP-210x
- AMS Gepard core

DSP address code optimization

variable set: { a, b, c, d }

access sequence: b, d, a, c, d, a, c, b, a, d, a, c, d

*alphabetic
layout*

0	a
1	b
2	c
3	d

AR = 1
AR += 2
AR -= 3
AR += 2
AR ++
AR -= 3
AR += 2
AR --
AR --
AR += 3
AR -= 3
AR += 2
AR ++

cost: 9

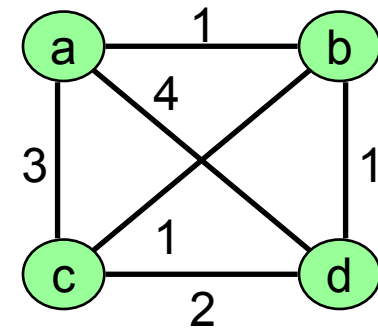
*optimized
layout*

0	c
1	a
2	d
3	b

AR = 3
AR --
AR --
AR --
AR += 2
AR --
AR --
AR --
AR += 3
AR -= 2
AR ++
AR --
AR --
AR += 2

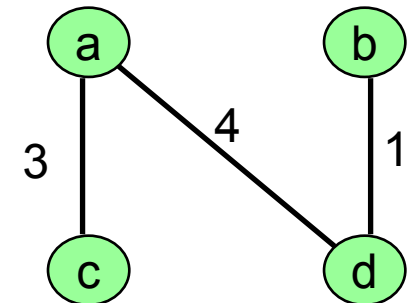
cost: 5

access graph:



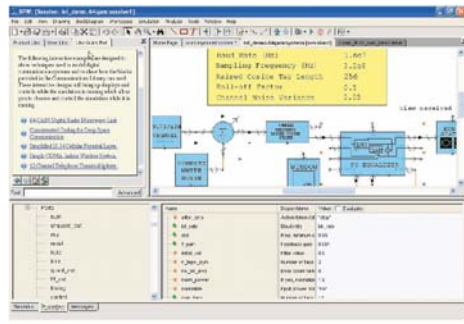
max.

Hamiltonian path:



www.address-code-optimization.org

Source-level code optimization

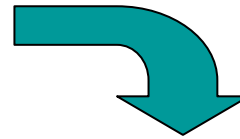


Algorithm design

- better performance
- lower code size
- highly reusable

[illegible]

C code generation



Poor code quality!



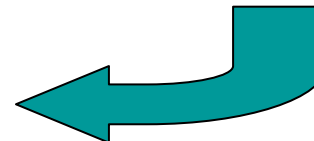
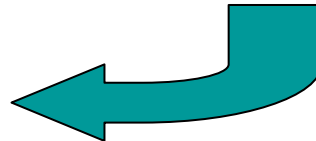
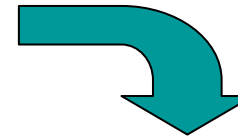
Compilation to assembly

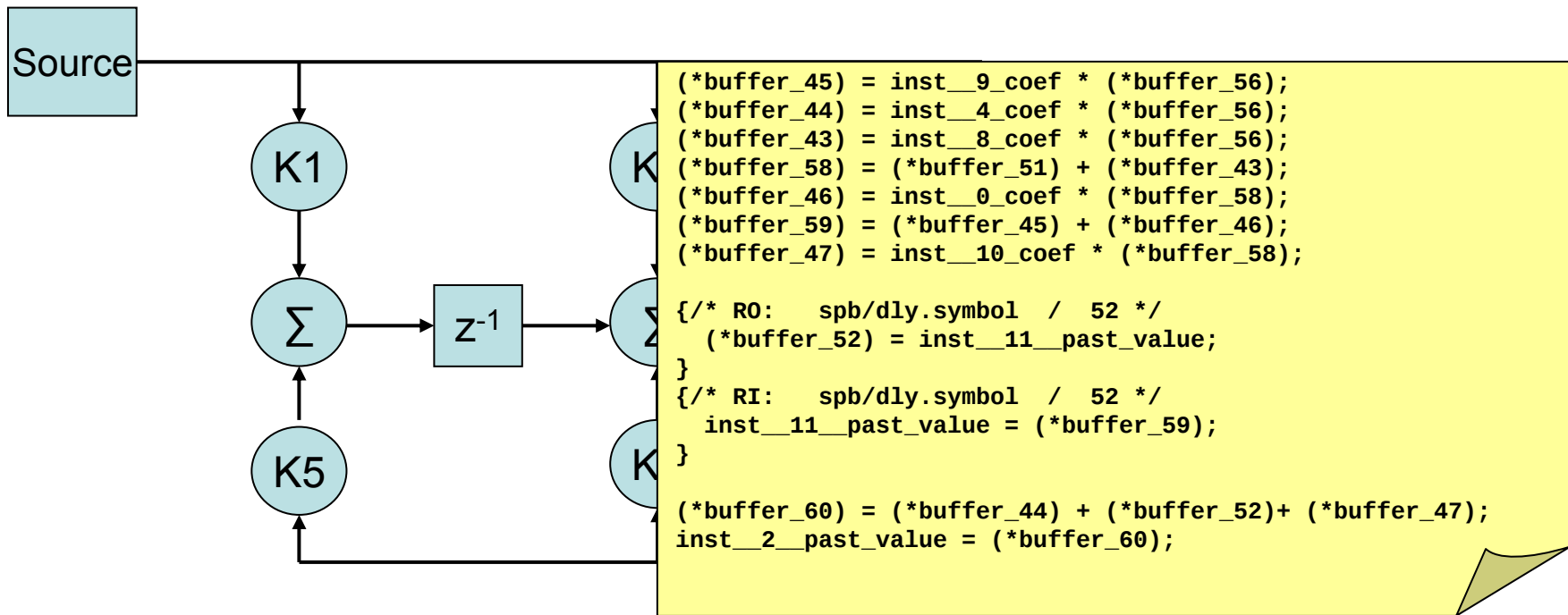


„Software washing“



„Software washing machine“

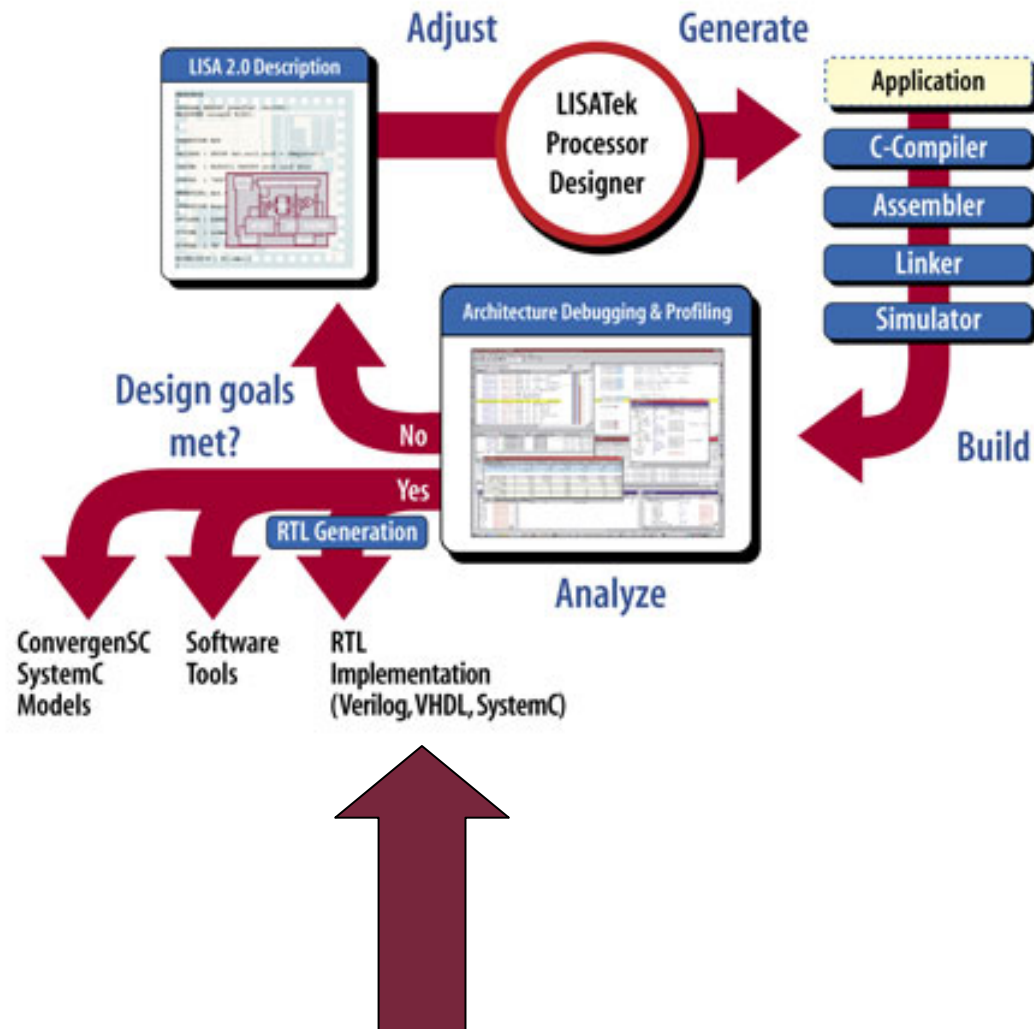


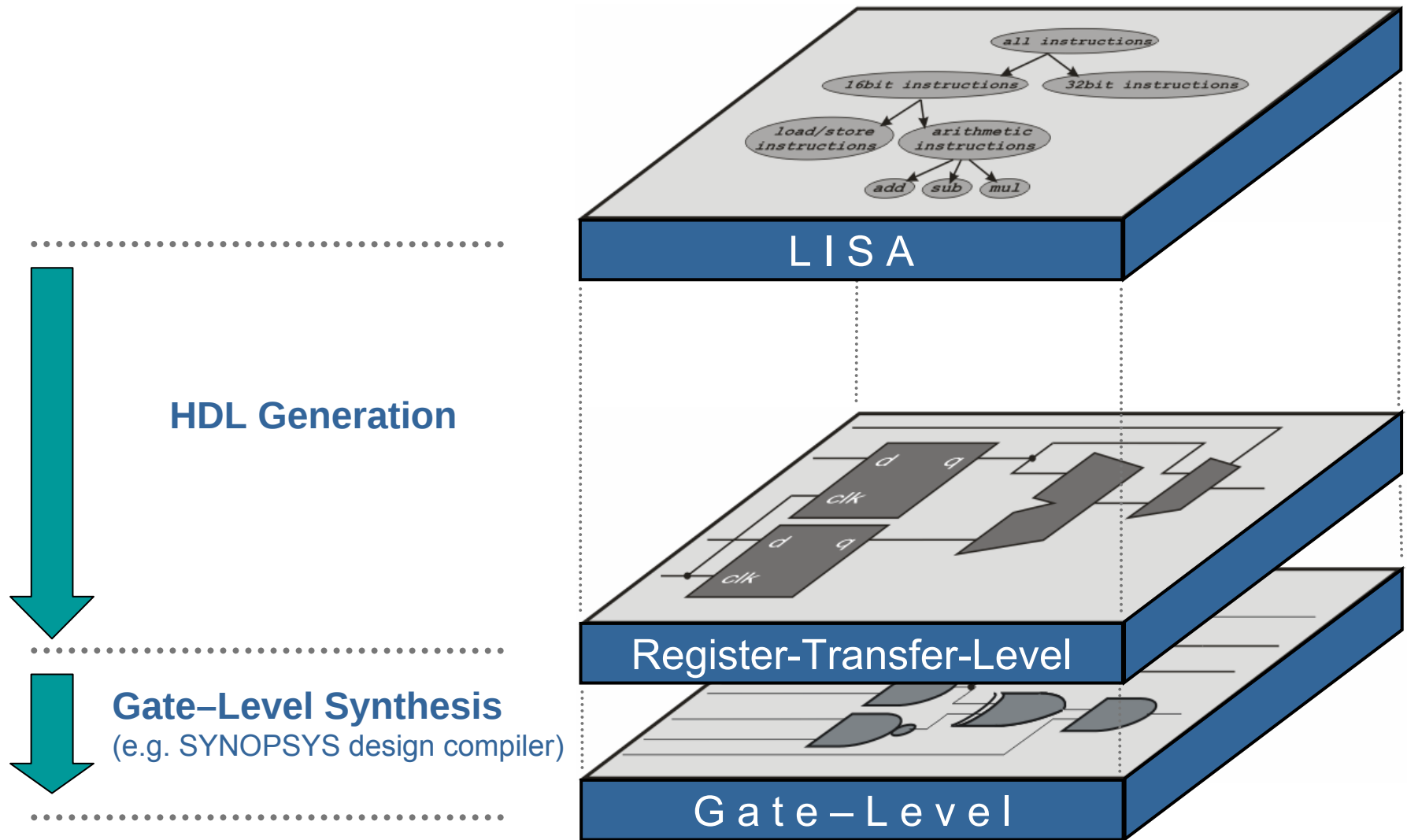


- Very conservative, block-oriented code generation scheme
- Even simple code transformations have great impact, e.g.
 - Variable localization
 - If-statement merging
 - Optimized data type selection

4. ASIP architecture design

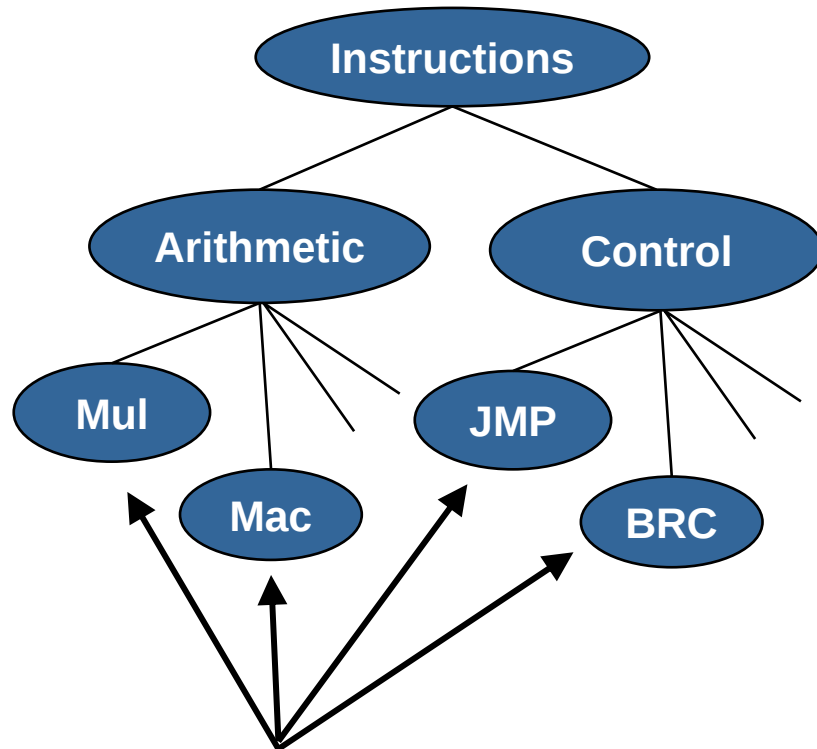
ASIP implementation after exploration





Challenges in Automated ASIP Implementation

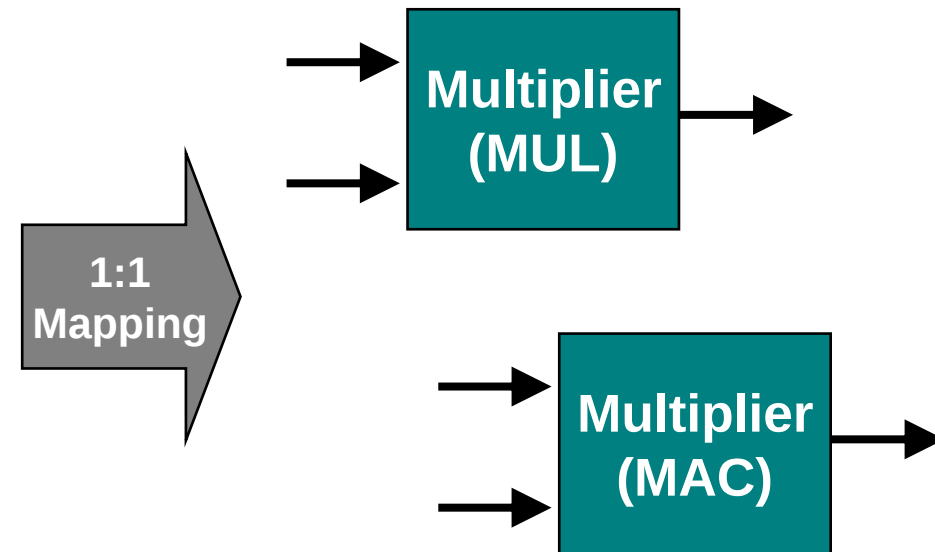
ADL:



Independent description of instruction behavior:

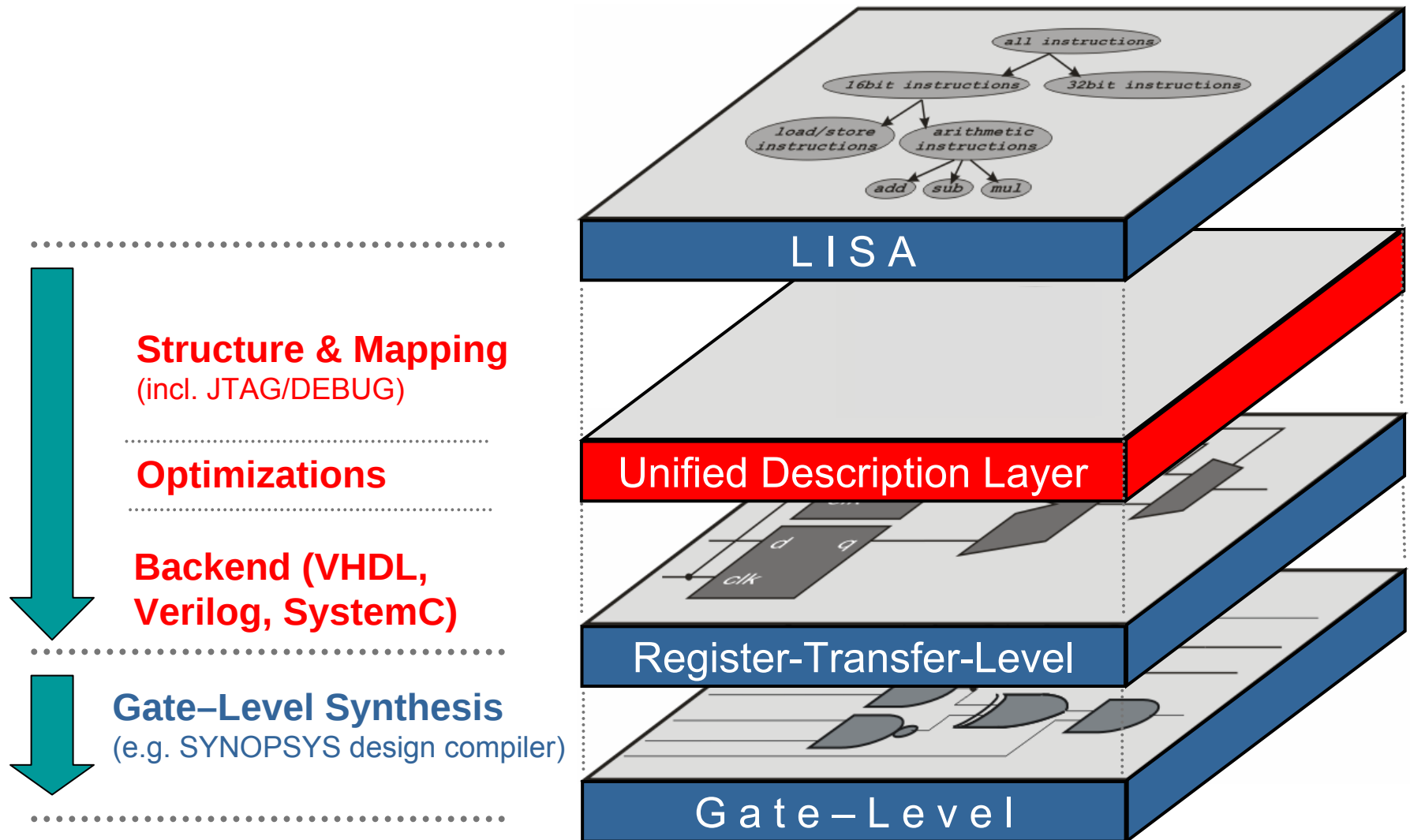
+ Efficient Design Space Exploration

HDL:



Independent mapping to hardware blocks:

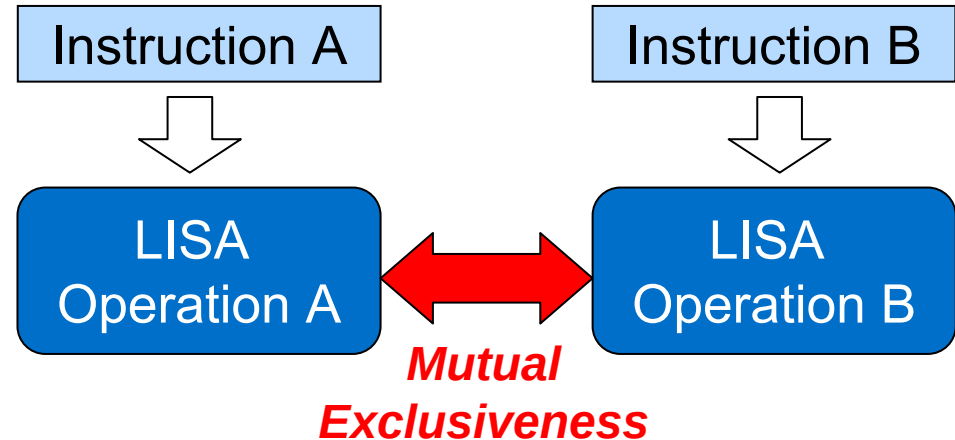
- Insufficient architectural efficiency by 1:1 mapping



Optimization strategies

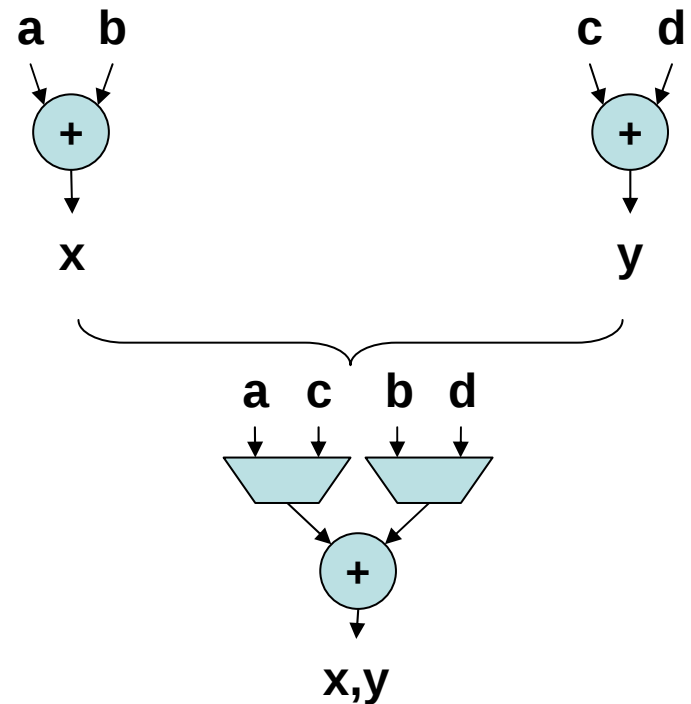
LISA: separate descriptions
for separate instructions

Goal: share hardware
for separate instructions



Possible Optimizations

- ALU Sharing



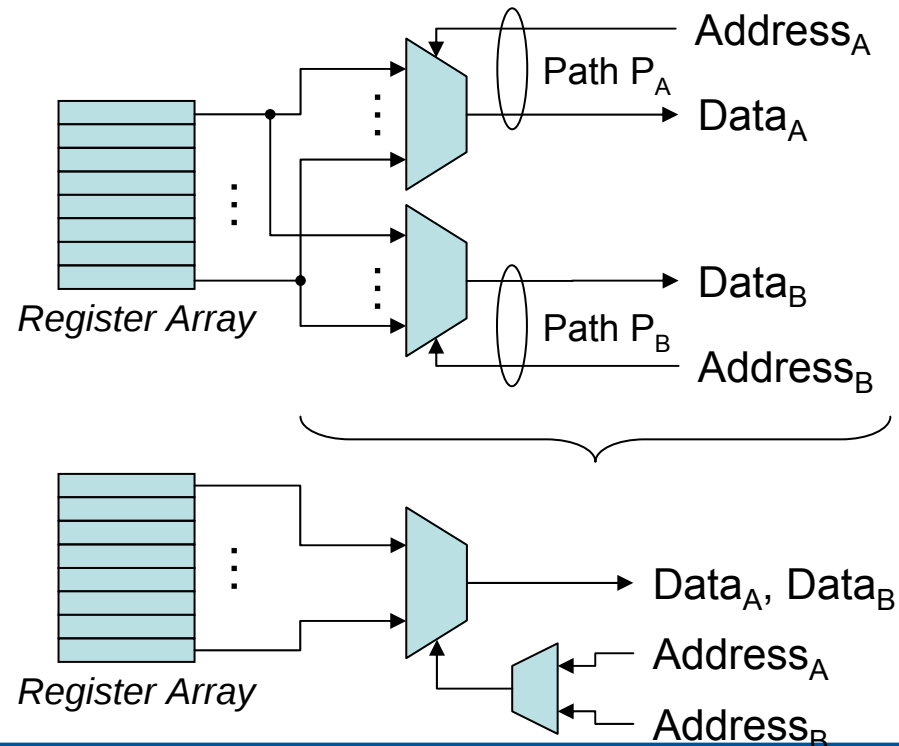
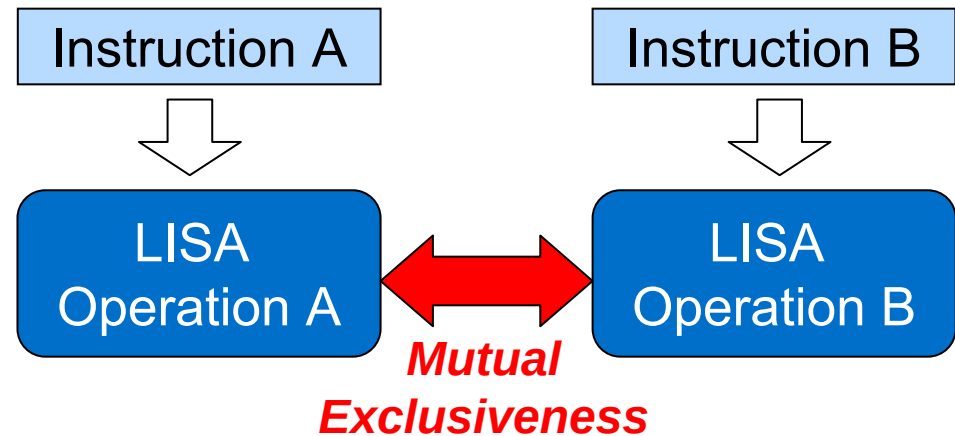
Optimization strategies

LISA: separate descriptions
for separate instructions

Goal: same hardware for
separate instructions

Possible Optimizations

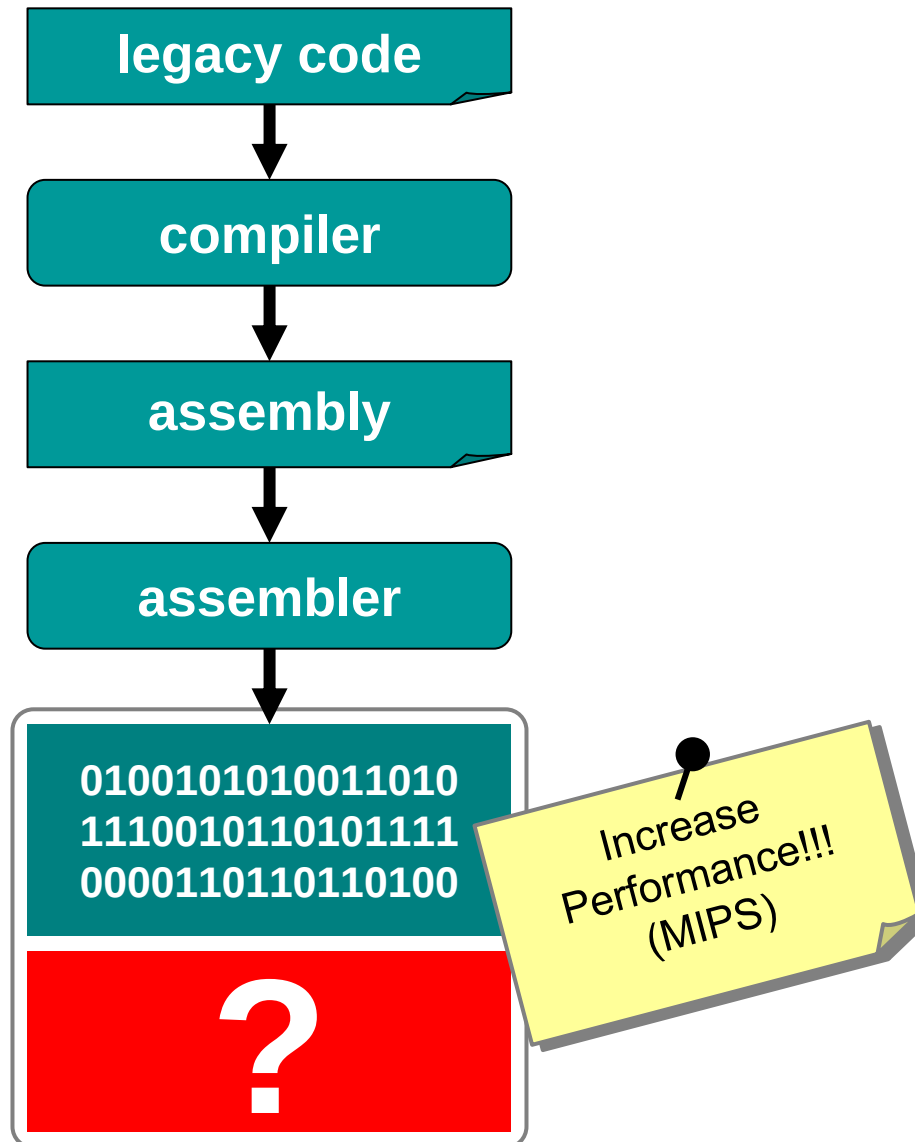
- ALU Sharing
 - Path Sharing
 - ...
- } *Resource Sharing*

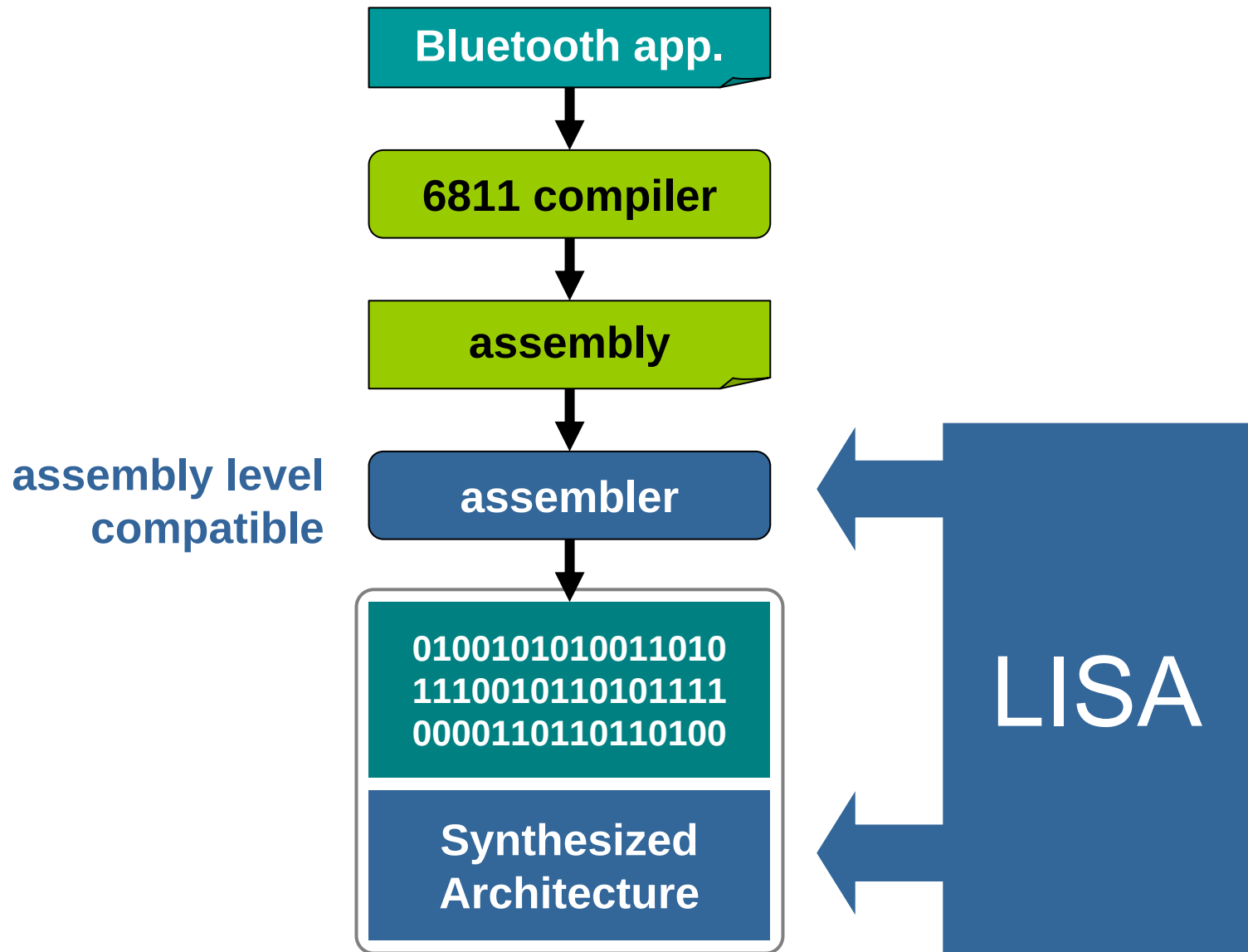


5. Case study

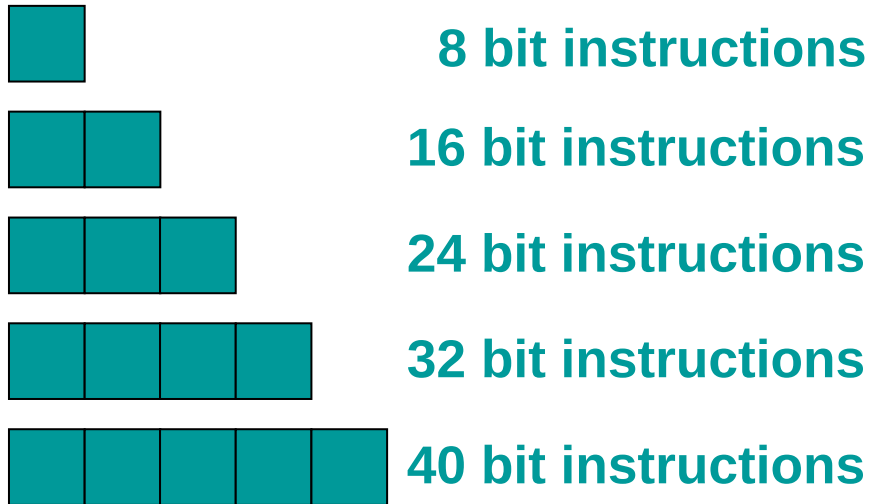
Project Goals:

- Performance (MIPS) must be increased
 - Compatibility on the assembly level
for reuse of legacy code
(Integration into existing tool flow)
 - Royalty free design
- ➔ compatible architecture developed with LISA
using RTL processor synthesis





original 6811 Processor



Instruction is fetched by 8 bit blocks:

→ up to **5 cycles** for fetching!



LISA 6811 Processor



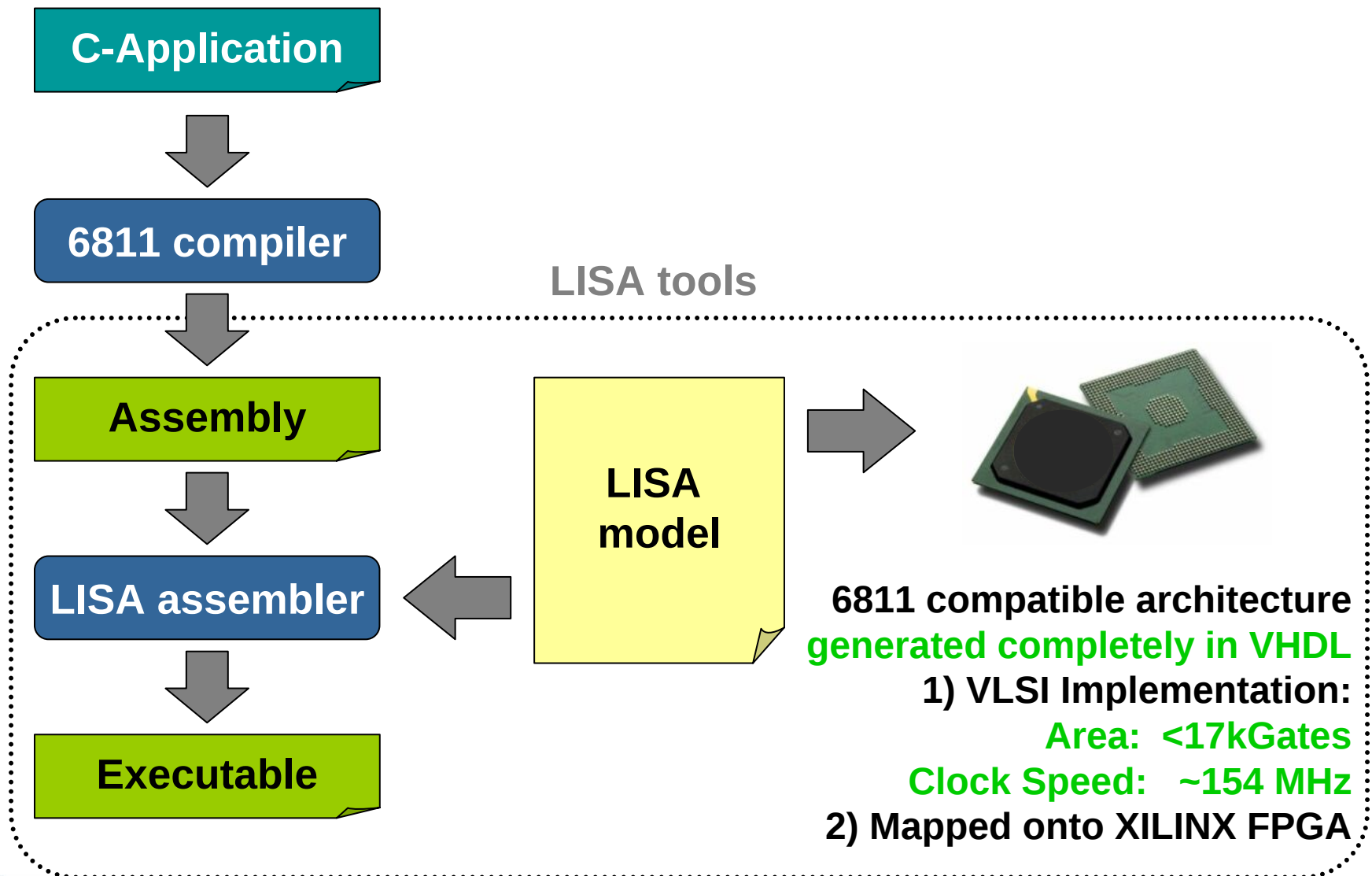
16 bit are fetched simultaneously:

→ **max 2 cycles** for fetching!

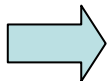
+ **pipelined** architecture

+ possibility for **special instructions**

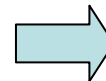
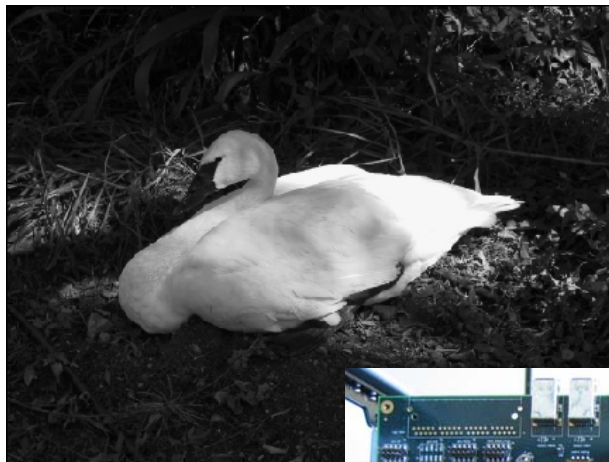
Tools Flow and RTL Processor Synthesis



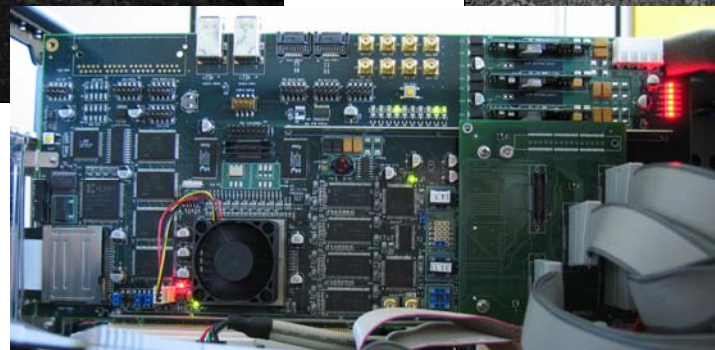
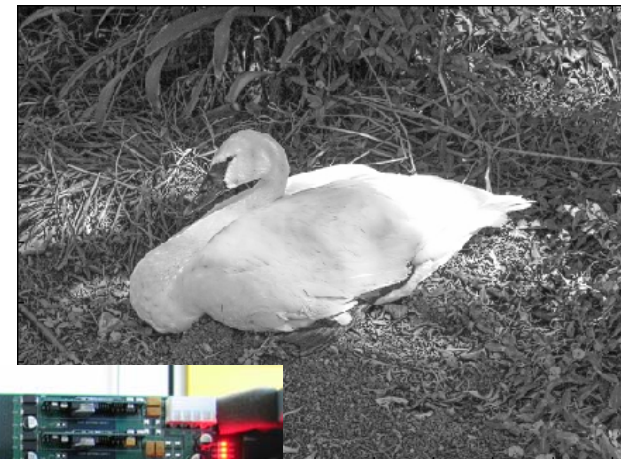
Retinex ASIP (digital image enhancement)



IN



OUT



Virtex II FPGA board

ASIC (0.18 μm):

- 102 kGates, 93 MHz

ASIP (0.13 μm):

- 124 kGates, 154 MHz
- SW programmable!

6. Advanced research topics

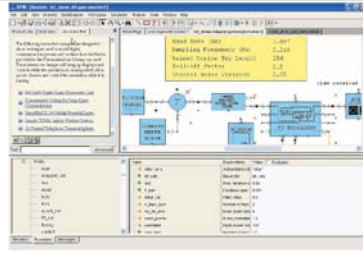
Generalized ASIP architecture design flow

Algorithm design
(Matlab, SPW, ...)

C code generation
or implementation

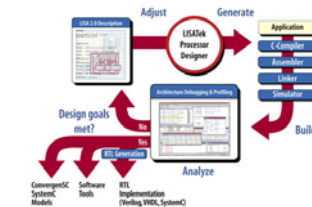
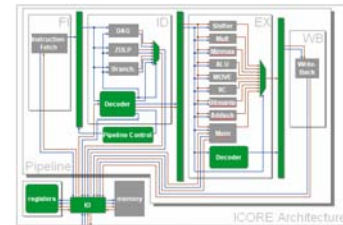
initial
architecture

architecture
optimization

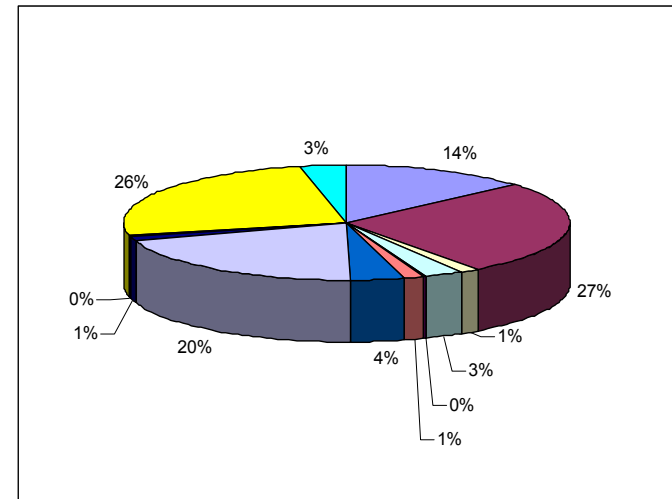


```

scale = 0.5;
for stage = 0: stageCount-1
    for LoopCount = 0: LoopCount-1
        LoopCount = LoopCount + 1;
        if LoopCount == 1
            F1 = input('Enter F1 value: ');
            F2 = input('Enter F2 value: ');
            F3 = input('Enter F3 value: ');
            F4 = input('Enter F4 value: ');
            F5 = input('Enter F5 value: ');
            F6 = input('Enter F6 value: ');
            F7 = input('Enter F7 value: ');
            F8 = input('Enter F8 value: ');
            F9 = input('Enter F9 value: ');
            F10 = input('Enter F10 value: ');
            F11 = input('Enter F11 value: ');
            F12 = input('Enter F12 value: ');
            F13 = input('Enter F13 value: ');
            F14 = input('Enter F14 value: ');
            F15 = input('Enter F15 value: ');
            F16 = input('Enter F16 value: ');
            F17 = input('Enter F17 value: ');
            F18 = input('Enter F18 value: ');
            F19 = input('Enter F19 value: ');
            F20 = input('Enter F20 value: ');
            F21 = input('Enter F21 value: ');
            F22 = input('Enter F22 value: ');
            F23 = input('Enter F23 value: ');
            F24 = input('Enter F24 value: ');
            F25 = input('Enter F25 value: ');
            F26 = input('Enter F26 value: ');
            F27 = input('Enter F27 value: ');
            F28 = input('Enter F28 value: ');
            F29 = input('Enter F29 value: ');
            F30 = input('Enter F30 value: ');
            F31 = input('Enter F31 value: ');
            F32 = input('Enter F32 value: ');
            F33 = input('Enter F33 value: ');
            F34 = input('Enter F34 value: ');
            F35 = input('Enter F35 value: ');
            F36 = input('Enter F36 value: ');
            F37 = input('Enter F37 value: ');
            F38 = input('Enter F38 value: ');
            F39 = input('Enter F39 value: ');
            F40 = input('Enter F40 value: ');
            F41 = input('Enter F41 value: ');
            F42 = input('Enter F42 value: ');
            F43 = input('Enter F43 value: ');
            F44 = input('Enter F44 value: ');
            F45 = input('Enter F45 value: ');
            F46 = input('Enter F46 value: ');
            F47 = input('Enter F47 value: ');
            F48 = input('Enter F48 value: ');
            F49 = input('Enter F49 value: ');
            F50 = input('Enter F50 value: ');
            F51 = input('Enter F51 value: ');
            F52 = input('Enter F52 value: ');
            F53 = input('Enter F53 value: ');
            F54 = input('Enter F54 value: ');
            F55 = input('Enter F55 value: ');
            F56 = input('Enter F56 value: ');
            F57 = input('Enter F57 value: ');
            F58 = input('Enter F58 value: ');
            F59 = input('Enter F59 value: ');
            F60 = input('Enter F60 value: ');
            F61 = input('Enter F61 value: ');
            F62 = input('Enter F62 value: ');
            F63 = input('Enter F63 value: ');
            F64 = input('Enter F64 value: ');
            F65 = input('Enter F65 value: ');
            F66 = input('Enter F66 value: ');
            F67 = input('Enter F67 value: ');
            F68 = input('Enter F68 value: ');
            F69 = input('Enter F69 value: ');
            F70 = input('Enter F70 value: ');
            F71 = input('Enter F71 value: ');
            F72 = input('Enter F72 value: ');
            F73 = input('Enter F73 value: ');
            F74 = input('Enter F74 value: ');
            F75 = input('Enter F75 value: ');
            F76 = input('Enter F76 value: ');
            F77 = input('Enter F77 value: ');
            F78 = input('Enter F78 value: ');
            F79 = input('Enter F79 value: ');
            F80 = input('Enter F80 value: ');
            F81 = input('Enter F81 value: ');
            F82 = input('Enter F82 value: ');
            F83 = input('Enter F83 value: ');
            F84 = input('Enter F84 value: ');
            F85 = input('Enter F85 value: ');
            F86 = input('Enter F86 value: ');
            F87 = input('Enter F87 value: ');
            F88 = input('Enter F88 value: ');
            F89 = input('Enter F89 value: ');
            F90 = input('Enter F90 value: ');
            F91 = input('Enter F91 value: ');
            F92 = input('Enter F92 value: ');
            F93 = input('Enter F93 value: ');
            F94 = input('Enter F94 value: ');
            F95 = input('Enter F95 value: ');
            F96 = input('Enter F96 value: ');
            F97 = input('Enter F97 value: ');
            F98 = input('Enter F98 value: ');
            F99 = input('Enter F99 value: ');
            F100 = input('Enter F100 value: ');
        end
        F1 = F1 * scale;
        F2 = F2 * scale;
        F3 = F3 * scale;
        F4 = F4 * scale;
        F5 = F5 * scale;
        F6 = F6 * scale;
        F7 = F7 * scale;
        F8 = F8 * scale;
        F9 = F9 * scale;
        F10 = F10 * scale;
        F11 = F11 * scale;
        F12 = F12 * scale;
        F13 = F13 * scale;
        F14 = F14 * scale;
        F15 = F15 * scale;
        F16 = F16 * scale;
        F17 = F17 * scale;
        F18 = F18 * scale;
        F19 = F19 * scale;
        F20 = F20 * scale;
        F21 = F21 * scale;
        F22 = F22 * scale;
        F23 = F23 * scale;
        F24 = F24 * scale;
        F25 = F25 * scale;
        F26 = F26 * scale;
        F27 = F27 * scale;
        F28 = F28 * scale;
        F29 = F29 * scale;
        F30 = F30 * scale;
        F31 = F31 * scale;
        F32 = F32 * scale;
        F33 = F33 * scale;
        F34 = F34 * scale;
        F35 = F35 * scale;
        F36 = F36 * scale;
        F37 = F37 * scale;
        F38 = F38 * scale;
        F39 = F39 * scale;
        F40 = F40 * scale;
        F41 = F41 * scale;
        F42 = F42 * scale;
        F43 = F43 * scale;
        F44 = F44 * scale;
        F45 = F45 * scale;
        F46 = F46 * scale;
        F47 = F47 * scale;
        F48 = F48 * scale;
        F49 = F49 * scale;
        F50 = F50 * scale;
        F51 = F51 * scale;
        F52 = F52 * scale;
        F53 = F53 * scale;
        F54 = F54 * scale;
        F55 = F55 * scale;
        F56 = F56 * scale;
        F57 = F57 * scale;
        F58 = F58 * scale;
        F59 = F59 * scale;
        F60 = F60 * scale;
        F61 = F61 * scale;
        F62 = F62 * scale;
        F63 = F63 * scale;
        F64 = F64 * scale;
        F65 = F65 * scale;
        F66 = F66 * scale;
        F67 = F67 * scale;
        F68 = F68 * scale;
        F69 = F69 * scale;
        F70 = F70 * scale;
        F71 = F71 * scale;
        F72 = F72 * scale;
        F73 = F73 * scale;
        F74 = F74 * scale;
        F75 = F75 * scale;
        F76 = F76 * scale;
        F77 = F77 * scale;
        F78 = F78 * scale;
        F79 = F79 * scale;
        F80 = F80 * scale;
        F81 = F81 * scale;
        F82 = F82 * scale;
        F83 = F83 * scale;
        F84 = F84 * scale;
        F85 = F85 * scale;
        F86 = F86 * scale;
        F87 = F87 * scale;
        F88 = F88 * scale;
        F89 = F89 * scale;
        F90 = F90 * scale;
        F91 = F91 * scale;
        F92 = F92 * scale;
        F93 = F93 * scale;
        F94 = F94 * scale;
        F95 = F95 * scale;
        F96 = F96 * scale;
        F97 = F97 * scale;
        F98 = F98 * scale;
        F99 = F99 * scale;
        F100 = F100 * scale;
    end
end
    
```



Profiling



A closer look at profilers: ASM level

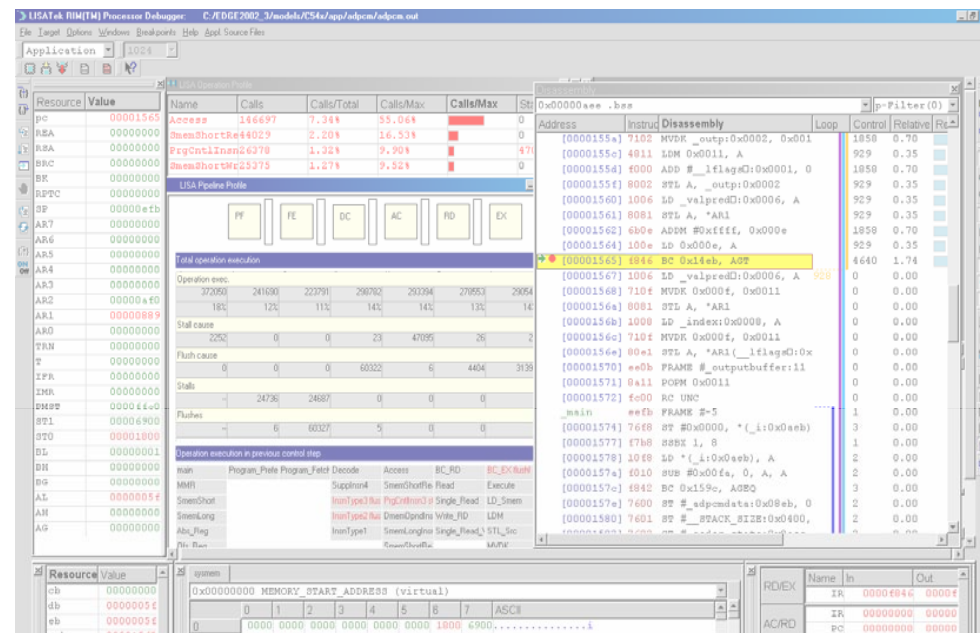
Algorithm design
(Matlab, SPW, ...)

C code generation
or implementation

initial
architecture

architecture
optimization

ASM level



- requires full architecture description
- slow (~1000x native C)

A closer look at profilers: source level

Algorithm design
(Matlab, SPW, ...)

C code generation
or implementation

initial
architecture

architecture
optimization

sample app: image corner detection



```
susan_corners(in,r,bp,max_no,corner_list,x_size,y_size)
uchar      *in, *bp;
int        *r, max_no, x_size, y_size;
CORNER_LIST corner_list;
{
int  n,x,y,sq,xx,yy,
    i,j,*cgx,*cgy;
float divide;
uchar c,*p,*cp;

memset (r,0,x_size * y_size * sizeof(int));

cgx=(int *)malloc(x_size*y_size*sizeof(int));
cgy=(int *)malloc(x_size*y_size*sizeof(int));

for (i=5;i<y_size-5;i++)
  for (j=5;j<x_size-5;j++) {
    n=100;
    p=in + (i-3)*x_size + j - 1;
    cp=bp + in[i*x_size+j];

    n+=*(cp-*p++);
    n+=*(cp-*p++);
    n+=*(cp-*p);
    p+=x_size-3;

    n+=*(cp-*p++);
    n+=*(cp-*p++);
    n+=*(cp-*p++);
    n+=*(cp-*p);
    p+=x_size-5;

    n+=*(cp-*p++);
    n+=*(cp-*p++);
    n+=*(cp-*p++);
    n+=*(cp-*p);
    p+=x_size-6;

    n+=*(cp-*p++);
    n+=*(cp-*p++);
    n+=*(cp-*p);
  }
}
```

ds.			
ls	self ms/call	total ms/call	name
1	270.00	270.00	susan_corners
3	0.00	0.00	getint
1	0.00	0.00	_GLOBAL__I_usage_GCOV
1	0.00	0.00	corner_draw
1	0.00	0.00	get_image
1	0.00	0.00	put_image
1	0.00	0.00	setup_brightness_lut

hot spot

Which instructions implement
this code efficiently?

Do not neglect compiler optimizations

Algorithm design
(Matlab, SPW, ...)

C code generation
or implementation

initial
architecture

architecture
optimization

```
while(corner_list[n].info != 7)
{
    if (drawing_mode==0)
    {
        p = in + (corner_list[n].y-1)*x_size + corner_list[n].x - 1;
        *p+=255; *p+=255; *p=255; p+=x_size-2;
        *p+=255; *p+=0; *p=255; p+=x_size-2;
        *p+=255; *p+=255; *p=255;
        n++;
    }
}
```

exec
count

gccv

```
372: 1447: while(corner_list[n].info != 7)
-: 1448: {
371: 1449:     if (drawing_mode==0)
-: 1450:     {
371: 1451:         p = in + (corner_list[n].y-1)*x_size + corner_list[n].x - 1;
371: 1452:         *p+=255; *p+=255; *p=255; p+=x_size-2;
371: 1453:         *p+=255; *p+=0; *p=255; p+=x_size-2;
371: 1454:         *p+=255; *p+=255; *p=255;
```

$p = in + (corner_list[n].y-1) * x_size + corner_list[n].x - 1;$

• 5x ADD
• 2x SUB
• 3x MUL

• 5x ADD
• 2x SUB
• 1x MUL

Wrong ISA decisions may result!

real code (optimized)

μ-profiling approach

```
372: 1447: while(corner_list[n].info != 7)
-: 1448: {
371: 1449:   if (drawing_mode==0)
-: 1450:   {
371: 1451:     p = in + (corner_list[n].y-1)*x_size + corner_list[n].x - 1;
371: 1452:     *p++=255; *p++=255; *p=255; p+=x_size-2;
371: 1453:     *p++=255; *p++=0; *p=255; p+=x_size-2;
371: 1454:     *p++=255; *p++=255; *p=255;
371: 1455:     n++;
-: 1456:   }
```

generate
low-level C code

```
t4692 = (char *)corner_list_1011;
t4691 = n_1016 * 24;
t4690 = t4692 + t4691;
t4693 = (struct lance_tag_13 *)t4690;
t4694 = (char *)t4693;
t4695 = t4694 + 4;
t4696 = (int *)t4695;
t4697 = *t4696 * x_size_1012;
t4698 = (char *)in_1010;
t4699 = t4697 * 1;
t4700 = t4698 + t4699;
t4701 = (unsigned char *)t4700;
t4704 = (char *)corner_list_1011;
t4703 = n_1016 * 24;
t4702 = t4704 + t4703;
t4705 = (struct lance_tag_13 *)t4702;
t4706 = (char *)t4705;
t4707 = t4706 + 0;
t4708 = (int *)t4707;
t4709 = (char *)t4701;
t4710 = *t4708 * 1;
t4711 = t4709 + t4710;
t4712 = (unsigned char *)t4711;
p_1015 = t4712;
```

perform
compiler
optimizations

```
t6531 = (char *)in_1010;
t192 = (char *)corner_list_1011;
t6533 = 4 + t192;
t6535 = t6533 + t191;
t6536 = (int *)t6535;
t6537 = *t6536 * x_size_1012;
t6538 = t6531 + t6537;
t6541 = t192 + t191;
t6542 = (int *)t6541;
t6543 = t6538 + *t6542;
t6544 = (unsigned char *)t6543;
```

instrument
code

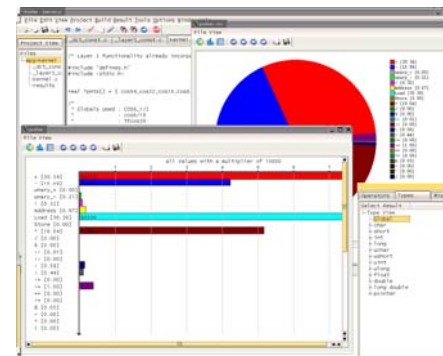
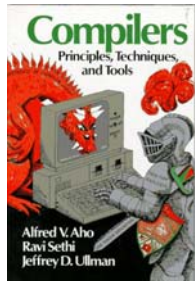
count ADD

count MUL

count LOAD

```
t6531 = (char *)in_1010;
t192 = (char *)corner_list_1011;
t6533 = 4 + t192;
t6535 = t6533 + t191;
t6536 = (int *)t6535;
t6537 = *t6536 * x_size_1012;
t6538 = t6531 + t6537;
t6541 = t192 + t191;
t6542 = (int *)t6541;
t6543 = t6538 + *t6542;
t6544 = (unsigned char *)t6543;
```

compile &
execute



	<i>C source level (e.g. gprof)</i>	<i>assembly level (e.g. LISATek)</i>	<i>Micro-profiler</i>
<i>primary application</i>	Source code optimization	ISA and architecture optimization	ISA and architecture optimization
<i>needs architectural details</i>	No	Yes	No
<i>speed</i>	High	Low	Medium
<i>Profiling granularity</i>	coarse	fine	fine

Micro-profiler in the design flow

➤ Precise profiling of:

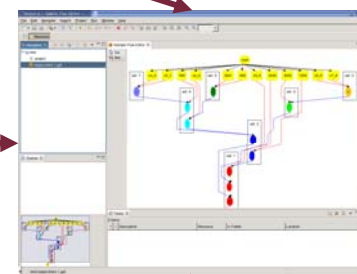
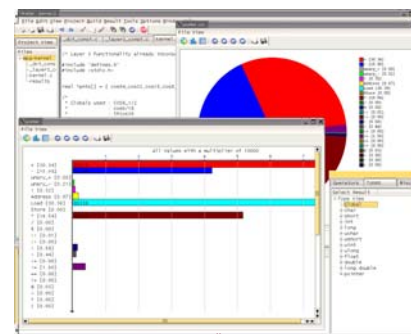
- Operator execution count
- Data type use
- Variable word lengths
- Memory access patterns

➤ Guide designer in basic architecture decisions:

- Inclusion of dedicated function units (floating point, address gen.)
- Leave out unused instructions
- Optimal data path word lengths
- Design of memory hierarchy (cache, scratchpad)

➤ Export profiling data for automatic ISA synthesis tool

- Effects of compiler optimizations predicted early



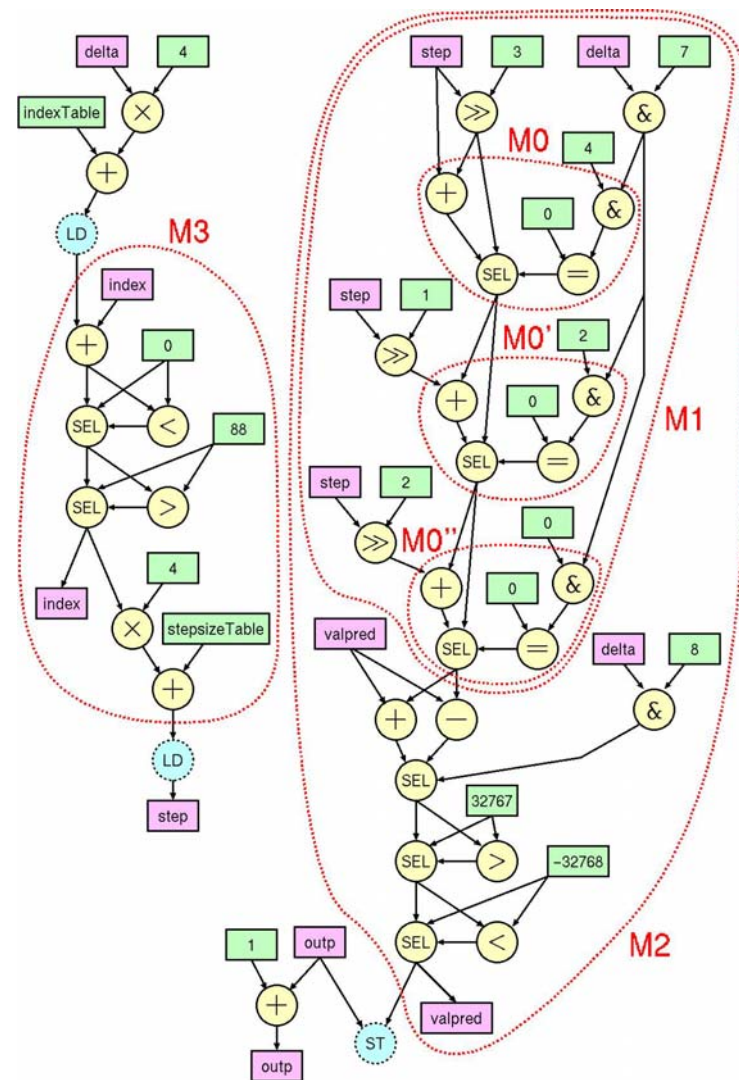
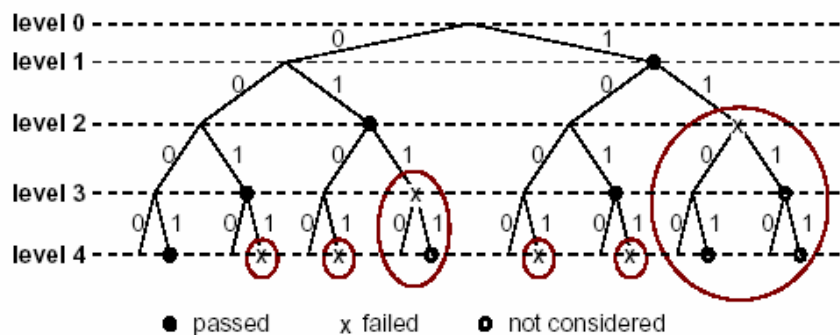
The screenshot shows a table with columns for 'Instruction', 'Count', 'Percentage', 'Word Length', and 'Data Type'. The table lists various instructions and their corresponding counts, percentages, word lengths, and data types. The table is sorted by 'Count' in descending order.

Instruction	Count	Percentage	Word Length	Data Type
Instruction 00000000	1000000	100.00%	32	Integer
Instruction 00000001	500000	50.00%	32	Integer
Instruction 00000002	250000	25.00%	32	Integer
Instruction 00000003	125000	12.50%	32	Integer
Instruction 00000004	62500	6.25%	32	Integer
Instruction 00000005	31250	3.12%	32	Integer
Instruction 00000006	15625	1.56%	32	Integer
Instruction 00000007	7812	0.78%	32	Integer
Instruction 00000008	3906	0.39%	32	Integer
Instruction 00000009	1953	0.19%	32	Integer
Instruction 0000000A	976	0.09%	32	Integer
Instruction 0000000B	488	0.04%	32	Integer
Instruction 0000000C	244	0.02%	32	Integer
Instruction 0000000D	122	0.01%	32	Integer
Instruction 0000000E	61	0.00%	32	Integer
Instruction 0000000F	30	0.00%	32	Integer

Custom instruction set synthesis: recent approaches

- J. Yang, C. Kyung et al.: *MetaCore: An Application Specific DSP Development System*, DAC 1998
- M. Gschwind: *Instruction Set Selection for ASIP Design*, CODES 1999
- K. Küçükçakar: *An ASIP Design Methodology for Embedded Systems*, CODES 1999
- H. Choi, C. Kyung et al.: *Synthesis of Application Specific Instructions for Embedded DSP Software*, IEEE Trans. CAD 1999
- F. Sun, S. Ravi et al.: *Synthesis of Custom Processors based on Extensible Platforms*, ICCAD 2002
- D. Goodwin, D. Petkov: *Automatic Generation of Application Specific Processors*, CASES 2003
- K. Atasu, L. Pozzi, P. Ienne: *Automatic Application-Specific Instruction-Set Extensions under Microarchitectural Constraints*, DAC 2003
- N. Clark, H. Zhong, S. Mahlke: *Processor Acceleration through Automated Instruction Set Customization*, MICRO 2003
- ... and many others

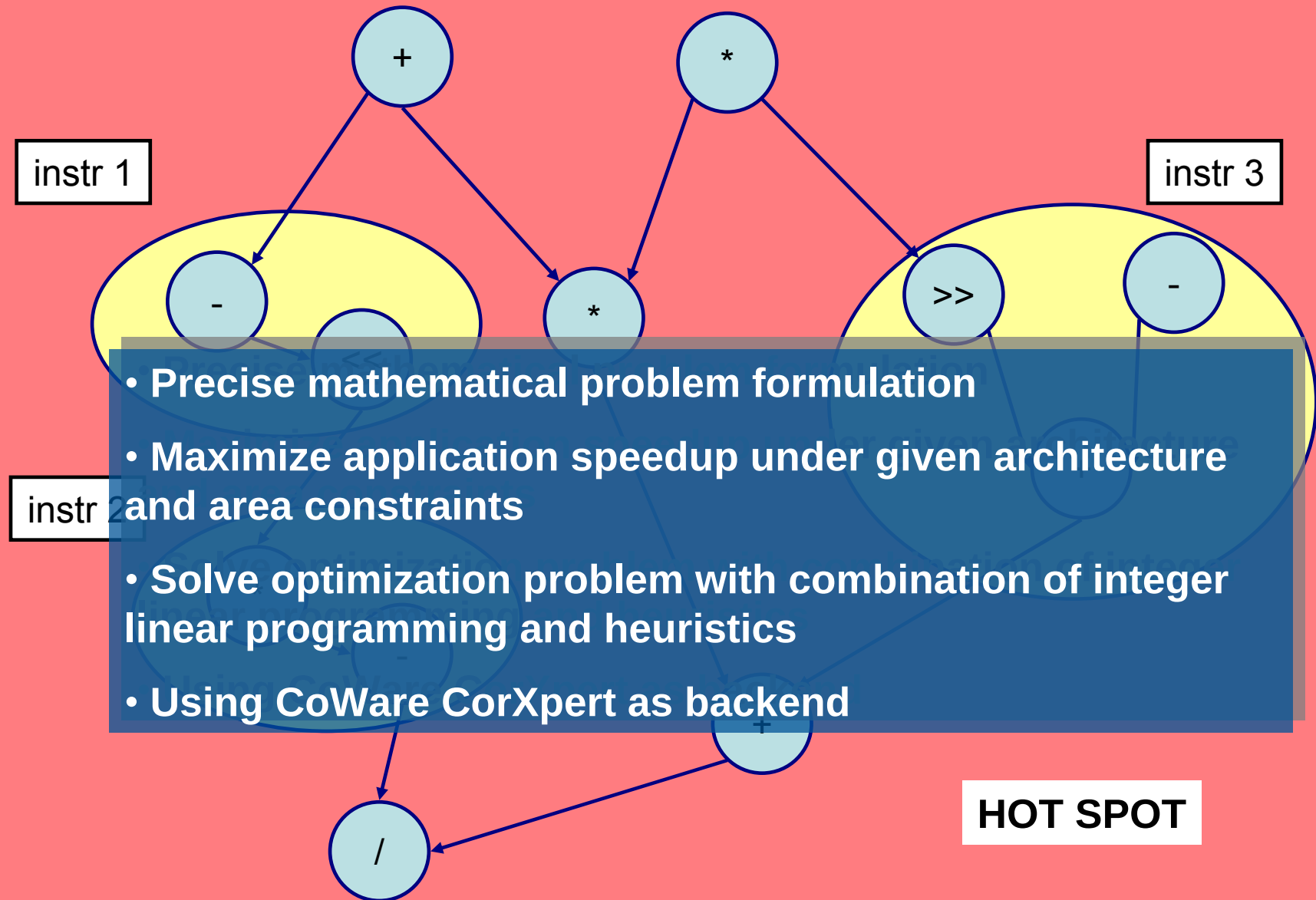
- Branch-and-Bound like algorithm for custom pattern identification under I/O constraints
- Capable of identifying optimal complex and disconnected patterns
- Requires fast and accurate cost estimation of speedup due to candidate patterns



Custom ISA synthesis: ISS approach

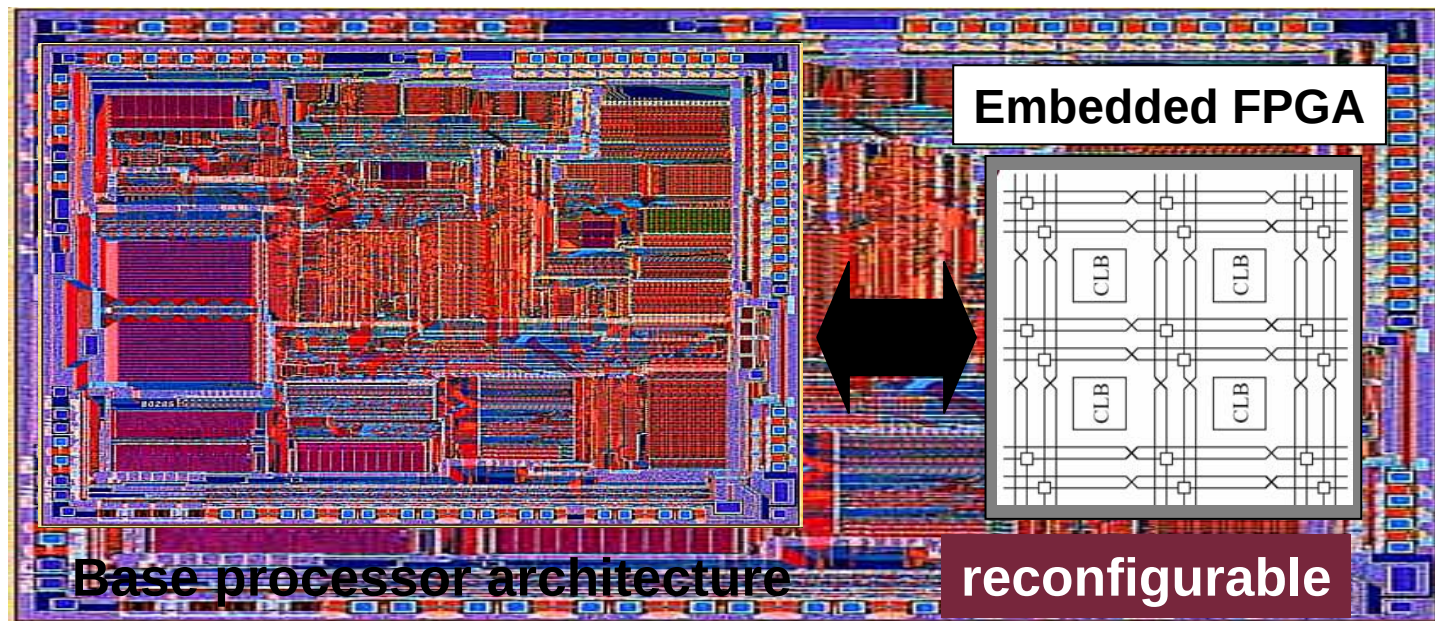


Abstract modeling: optimal data flow graph covering



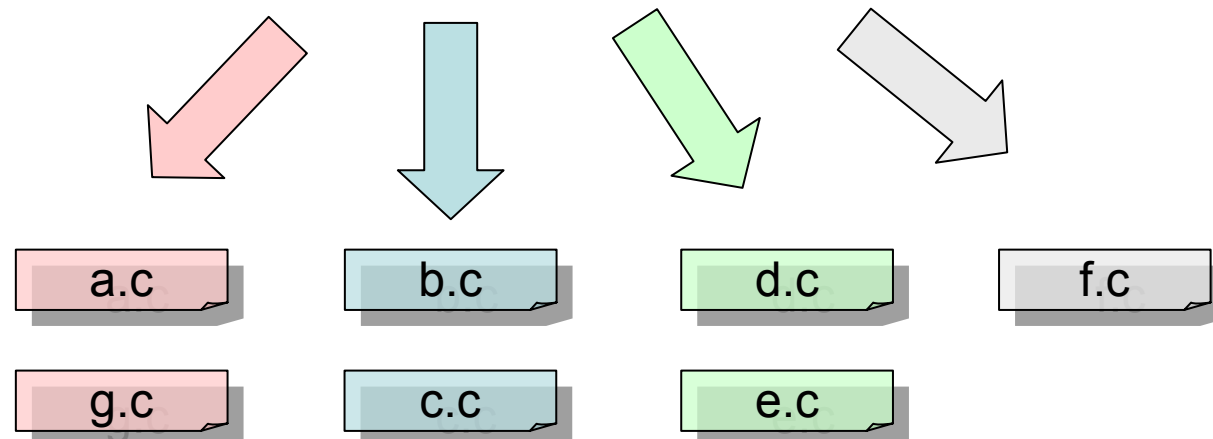
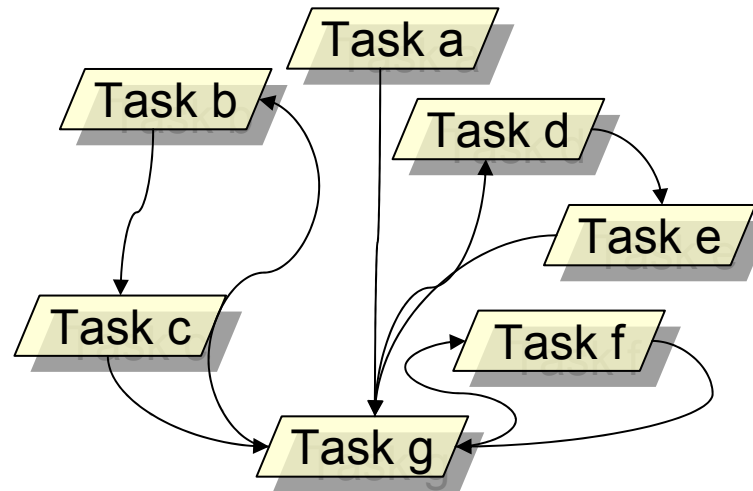
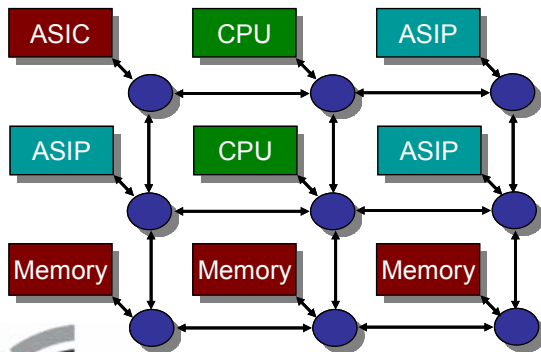
Outlook: reconfigurable ASIPs

- Currently: (one-time) configurable ASIPs
- Combination of ASIP and FPGA:
 - “field-configurable ASIPs”
 - Fast adaptations w/o HW modifications



Outlook: compilation for heterogeneous MPSoC's

- Urgently needed, but virtually not present today
- No tools even for simplest platforms (e.g. TI OMAP)
- Need for automated spatial and temporal task-to-processor mapping



- R. Leupers: *Code Optimization Techniques for Embedded Processors - Methods, Algorithms, and Tools*, Kluwer, 2000
- R. Leupers, P. Marwedel: *Retargetable Compiler Technology for Embedded Systems - Tools and Applications*, Kluwer, 2001
- A. Hoffmann, H. Meyr, R. Leupers: *Architecture Exploration for Embedded Processors with LISA*, Kluwer, 2002
- C. Rowen, S. Leibson: *Engineering the Complex SoC: Fast, Flexible Design with Configurable Processors*, Prentice Hall, 2004
- M. Gries, K. Keutzer, et al.: *Building ASIPs: The Mescal Methodology*, Springer, 2005
- P. lenne, R. Leupers (eds.): *Customizable and Configurable Embedded Processor Cores*, Morgan Kaufmann, to appear 2006

Thank you !

